UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

TÍTULO: Uso de dispositivos GPS e IMU para analizar la performance de deportistas de alto rendimiento

AUTORES: Barreto, Cristian y Robles, Lorena Belén

DIRECTOR: Fava, Laura

CODIRECTOR: Vilches, Diego

ASESOR PROFESIONAL: -

CARRERA: Licenciatura en Sistemas

Resumen

Dado la constante evolución de las tecnologías IoT y la aplicación de éstas en el competitivo estudio del mundo del deporte, se presenta en esta tesina el diseño y desarrollo de un sistema de monitoreo en tiempo real en el que se visualizan ciertas métricas logradas tras el procesamiento de los datos obtenidos por parte del dispositivo diseñado para el trabajo. Se detallan los filtros y cálculos matemáticos sobre los datos, el protocolo de comunicación utilizado entre el dispositivo y el ente receptor, la tecnología de almacenamiento utilizada y el software de monitoreo en cuestión. A su vez, se realiza una breve comparación con uno de los dispositivos de análisis de deportistas más ampliamente usado en la actualidad por equipos profesionales de distintas disciplinas, el Catapult Playertek.

Palabras Clave

Internet de las Cosas (IoT), electrónica, sensores, IMU, GPS, filtros de señales, eficiencia en deporte, protocolos de comunicación, MQTT, monitoreo en tiempo real, time series databases, dashboards, Raspberry Pi, Playertek.

Trabajos Realizados

Se integró una placa de desarrollo con un GPS y un IMU. Se procesaron los datos de los sensores para obtener velocidad y aceleración. Se comparó el rendimiento de filtros matemáticos para la fusión de los datos. Se realizó la conexión entre los dispositivos y un Broker MQTT. Se creó un script Python que procesa, formatea y almacena los datos en una TSDB. Se configuraron dashbords y paneles para visualizar las métricas. Se configuró una RPi 4 que actúa como nodo middleware y ejecuta las partes que conforman el sistema. Se realizó una sesión de entrenamiento como caso de prueba para demostrar el funcionamiento integral del trabajo en detalle.

Conclusiones

Teniendo en cuenta los resultados obtenidos, es posible afirmar que como primer prototipo se han logrado datos y métricas suficientemente confiables como para continuar incrementando su desarrollo, y de esta forma, competir directamente con los dispositivos de alto valor monetario ya existentes en el mercado. Se cree además, que este mismo trabajo llevado a cabo con dispositivos de hardware de mayor calidad haría que las métricas obtenidas sean aún más precisas, permitiendo hacer un análisis de los datos con mayor certeza y seguridad que ayudarán a tomar decisiones más fiables en el mundo del deporte competitivo.

Trabajos Futuros

Reemplazar el microcontrolador utilizado por uno multinúcleo, con la finalidad de ser capaz de paralelizar en hilos los módulos de polling del GPS y del IMU y así calcular de manera más precisa y eficiente las integrales en el tiempo transcurrido de muestreo. Implementar el filtro de Kalman, que permite la fusión, corrección y predicción de estados futuros de los datos de entrada, aumentando notoriamente la precisión de los datos obtenidos. En cuanto a la visualización de las métricas, puede desarrollarse un software propio y a medida que reemplace el utilizado, brindando mayor identidad y elegancia al único punto de entrada visual e interactivo de la aplicación.

Fecha de la presentación: NOVIEMBRE 2020

Universidad Nacional de La Plata

Facultad de Informática

Tesina de grado

Uso de dispositivos GPS e IMU para analizar la performance de deportistas de alto rendimiento

Autores:
Lorena Belén Robles
Cristian Barreto

Director:
Laura Fava
Co-Director:
Diego Vilches

Noviembre 2020

Índice general

Capítulo 1

Objetivos

1.1. Motivación

A lo largo de los últimos años, alrededor del mundo se ha comenzado a hablar de la *Cuarta revolución industrial* o *Industria* 4,0, entendiendo como tal al período en el cual las fronteras entre el mundo físico, digital y biológico comienzan a desintegrarse. Esta revolución de a poco comenzó a incursionarse en el área del deporte, donde las exigencias en las distintas disciplinas son cada vez mayores, ocasionando que se deba comenzar a evaluar e incorporar nuevas herramientas y tecnologías para obtener datos cada vez más precisos con el fin de mejorar el rendimiento deportivo ya sea de manera individual o colectiva. A su vez, la incorporación de estas herramientas provee mejoras en el estudio de lesiones pudiendo prevenirlas con mayor anticipación, mejoras en el panorama táctico y estratégico y en todo aspecto que conlleve a obtener información cada vez más precisa para su posterior estudio, brindando la posibilidad de abordar nuevos desafíos deportivos que en el transcurso de las últimas décadas se ha convertido en un mercado cada vez más competitivo.

Sabemos que la tecnología crece de manera exponencial y actualmente resulta prácticamente imposible desvincularlo de las actividades deportivas, llevando a que en los últimos años aparezca el término de *wearable technology*, dispositivos para vestir que vienen equipados con sensores como GPS, acelerómetros, giroscopios y magnetómetros. La manifestación de estos dispositivo hizo que los mejores equipos

de distintas disciplinas del deporte comiencen a interesarse e incorporarlos en sus primeras filas y a su vez, incluir analistas de datos en el equipo que sean capaces de estudiar estos datos para sacar provecho de ellos.

El problema de estos dispositivos es su elevado precio de compra y que son productos privativos, lo que impacta directamente en el presupuesto de equipos de menores recursos. De misma manera, al ser productos de naturaleza restrictiva también impacta en el sector académico y en el interés de los estudiantes, ya que dificulta el estudio del código fuente y su implementación. Partiendo del enfoque planteado anteriormente es posible aplicar el concepto de *Internet of the Things* (*IoT*) o *Internet de las cosas* a soluciones para el deporte.

Es por ello que para esta tesina, se propone desarrollar un sistema que trate de igualar el funcionamiento y la utilidad de estos productos comerciales para el mundo del deporte, ensamblando un dispositivo equipado con una placa de desarrollo y sensores económicos y accesibles por la comunidad, para posteriormente volcar los datos resultantes en gráficos que servirán como objeto de estudio para el cuerpo técnico de un equipo, entrenadores, o cualquier persona que esté interesada en el funcionamiento de estas tecnologías.

1.2. Objetivos generales

La presente tesina tiene como objetivo la recopilación de datos en tiempo real provenientes de múltiples dispositivos equipados con sensores como GPS, acelerómetro, giroscopio y magnetómetro, con el fin de tratarlos y procesarlos mediante algoritmos de Fusión de Sensores para transformar los datos crudos en información o variables de interés tales como velocidad, aceleración, distancia, sprints, cambios de dirección, entre otras.

Asimismo, estos datos serán almacenados en una base de datos y visualizados en una herramienta web mediante el uso de diversos paneles y gráficos acordes a cada medición, permitiendo a los entrenadores estudiar esta información en tiempo real brindando una amplia ventaja en el análisis de entrenamientos y competencias, con el fin de mejorar la planificación táctica y estratégica de equipos y deportistas de

alto rendimiento.

1.3. Objetivos específicos

A continuación se enumeran los puntos específicos a realizar para completar el desarrollo de esta tesina, analizando siempre la mejor opción para cumplir con los objetivos:

- Integrar la placa de desarrollo con los distintos sensores (GPS, acelerómetro, giroscopio y magnetómetro) para recopilar los datos.
- Procesar parte de los datos en los dispositivos para transformar los datos crudos en variables de interés tal como velocidad o aceleración.
- Lograr el envío de información desde los dispositivos hacia un broker utilizando el protocolo MQTT.
- Configurar una Raspberry Pi 4 que actuará como broker o nodo central de procesamiento y almacenamiento.
- Convertir los datos recibidos por el broker en datos con el formato adecuado para ser almacenados una base de datos de series temporales.
- Comparar la eficiencia y rendimiento de los distintos filtros matemáticos para la fusión de datos provenientes de los sensores.
- Visualizar los datos obtenidos en paneles de un sistema web.
- Organizar adecuadamente los paneles de visualización para lograr la mejor experiencia de usuario de los entrenadores.
- Desarrollar un caso de prueba que evalúe la usabilidad y utilidad de la aplicación.

Capítulo 2

Introducción

2.1. Internet de las cosas

Internet de las cosas o *Internet of the things* (*IoT*) es un concepto que surgió en el año 1999 en el Massachusetts Institute of Technology (MIT) y que refiere a la interconexión de dispositivos digitales a internet a través de un único identificador (UIDs), y a la habilidad de transferir datos a través de la red sin la necesidad de intervención humana.

Este concepto ha evolucionado con el avance de múltiples tecnologías, por ejemplo el análisis en tiempo real, el aprendizaje automático, el crecimiento de los sensores y de los sistemas integrados, las redes inalámbricas, los sistemas de control, etc. En el mercado de consumo, IoT es sinónimo de productos relacionados con el concepto de *smart houses* (casas inteligentes), que abarca una amplia gama de dispositivo y electrodomésticos, como accesorios de iluminación, termostatos, sistemas de seguridad para el hogar y cámaras, que pueden controlarse a través de dispositivos como Smartphones y altavoces inteligentes [1].

2.1.1. Aplicaciones

El amplio conjunto de aplicaciones para dispositivos IoT es comúnmente dividido en: espacios de consumo, comerciales, industriales, infraestructura, militares, y deporte [2], [3].

2.1.1.1. Aplicaciones de consumo

Una gran parte de los dispositivos IoT se crean para el uso del consumidor, incluido los vehículos conectados, la automatización del hogar, la tecnología *wearable* o "vestible", la salud conectada, y los dispositivos con capacidades de monitoreo remoto.

- **Smart Home**: los dispositivos IoT forman parte de la domótica, donde se incluyen artefactos de iluminación, calefacción, aire acondicionado, sistemas de seguridad, etc. Los beneficios a largo plazo podrían incluir ahorros de energía al garantizar que las luces y los dispositivos electrónicos se apaguen automáticamente cuando ya no sean requeridos. Un hogar inteligente podría basarse en una plataforma que controle todos los dispositivos inteligentes de la casa. Por ejemplo, el existe un HomeKit de Apple, donde existen productos para el hogar y estos pueden controlarse por una aplicación en iOS desde cualquier iPhone e incluso desde el Apple Watch. A su vez, los dispositivos IoT de la casa pueden controlarse por voz a través de Siri (el asistente virtual de Apple que permite hacer consultas a través de la voz y el lenguaje natural). Otro ejemplo es el caso del Smart Home Essentials de Lenovo, que es una línea de dispositivos domésticos inteligentes que se controlan a través de la aplicación Home de Apple o Siri sin la necesidad de un puente Wi-Fi. También hay centros dedicados para el hogar inteligente que se ofrecen como plataformas independientes para conectar diferentes productos para el hogar inteligente, estos son: Amazon Echo, Google Home y Samsung SmartThings Hub. Además de estos productos comerciales, existen otros ecosistemas de código abierto no patentados como Home Assistant, OpenHAB y Domoticz.
- **Cuidado de ancianos**: otra aplicación de un hogar inteligente es proporcio-

nar asistencia a personas con discapacidades y personas mayores, en el que el sistema doméstico se adapta a las discapacidades específicas del propietario. Algo de mucha utilidad es el control por voz, que ayuda a los usuarios que poseen limitaciones de visión y movilidad, así como también existen sistemas de alerta que se conectan a implantes auditivos para las personas con discapacidades auditivas. Existen también, otras aplicaciones como sensores que controlan emergencias médicas ante caídas o convulsiones. La tecnología de hogares inteligentes aplicada a estos escenarios pueden proveer a los usuarios más libertad y una mejor calidad de vida.

2.1.1.2. Aplicaciones comerciales

- **Médico y sanitario**: El internet de las cosas médicas (IoMT) se basa en la utilidad de los dispositivos IoT aplicados al área de la salud, con el fin de recopilar e investigar datos que servirán para monitorear comportamientos de las personas. Los dispositivos IoT se pueden usar para habilitar el monitoreo remoto de salud y sistema de notificaciones de emergencia. Estos dispositivos pueden ser monitores de presión arterial y frecuencia cardíaca, implantes, marcapasos, etc. Algunos hospitales empezaron a implementar "smart beds" o camas inteligentes que pueden detectar cuando un paciente está recostado y cuando intenta levantarse o ajustar la posición de la cama sin la interacción manual de las enfermeras.

- **Transporte**: El IoT puede resultar muy favorable en la comunicación y control de los transportes, conductores e infraestructura en general. La interacción dinámica entre estos actores dentro de un sistema de transporte permite: comunicación entre vehículos así como también dentro del mismo, control de tráfico inteligente, estacionamiento inteligente, sistemas electrónicos de cobro de peaje, logística, seguridad y asistencia en la carretera, entre las más importantes. Los sensores GPS, sensores de humedad y temperatura, envían datos a la plataforma IoT para que sean analizados y posteriormente enviados a los usuarios. De esta forma, los usuarios pueden rastrear en tiempo real los vehículos y pueden tomar decisiones apropiadas, y combinado con el aprendizaje automático también ayuda a reducir los accidentes de tránsito a la hora de introducir alertas de somnolencia o desatención (evaluando ciertos patrones

anómalos en la conducción) e incluso proporcionando vehículos sin conductor, como es el caso de Tesla, Inc.

- **Construcción y domótica**: Los dispositivos IoT se pueden usar para monitorear y controlar los sistemas mecánicos y eléctricos, en sistemas de automatización de hogar y también automatización de edificios [4]. En este contexto, las tres principales áreas son:

    - La integración de internet con los sistemas de gestión y regulación energética instalados en los edificios para crear "edificios inteligentes" que sean eficientes en el consumo de energía, impulsados por IoT.
    - El monitoreo en tiempo real para reducir el consumo de energía monitoreando el consumo de los ocupantes del edificio.
    - La integración de dispositivos inteligentes en el edificio y saber cómo podrían ser utilizados en distintas futuras aplicaciones.

2.1.1.3. Aplicaciones industriales

También conocidos como IIoT, los dispositivos industriales adquieren y analizan datos de equipos conectados, tecnología operativa, ubicaciones y personas que combinado con dispositivos de monitoreo ayuda a regular y controlar sistemas industriales [5].

- **Fabricación**: El IoT puede realizar la integración perfecta de varios dispositivos de fabricación equipados con diferentes capacidades de detección, identificación, procesamiento, comunicación, y conexión de red, abriendo la puerta para crear nuevas oportunidades comerciales de fabricación, gestión de equipos de fabricación junto con su proceso industrial, y la fabricación inteligente. Los sistemas inteligentes de IoT permiten la fabricación rápida de nuevos productos, la respuesta dinámica a las demandas de los productos y la optimización en tiempo real de la producción de las industrias y las redes de cadenas de suministros, mediante la interconexión de maquinaria, sensores y sistemas de control. Los sistemas de control digital para automatizar los controles de proceso, las herramientas de los operadores, y los sistemas de información para optimizar la seguridad de la planta están dentro del alcance

del IoT, pero también se extiende a la gestión de activos a través del mantenimiento predictivo, la evaluación estadística y las mediciones para maximizar la fiabilidad. Los sistemas de gestión industrial también se pueden integrar con redes inteligentes, lo que permite la optimización de energía en tiempo real mediante el uso de sensores. El IIoT podría estar generando tanto valor industrial que ya se ha comenzado a hablar de la cuarta revolución industrial, también conocida como industria 4.0.

- **Agricultura**: El IoT nos brinda una gran cantidad de aplicaciones para la agricultura, como puede ser la recopilación de datos sobre factores del clima como temperatura, humedad, lluvia, velocidad del viento, nutrientes de la tierra, etc. Estos datos se pueden usar para automatizar las técnicas agrícolas, tomar mejores decisiones para mejorar la calidad y la cantidad de las cosechas, minimizar el riesgo y desperdicio y reducir el esfuerzo requerido para administrar los cultivos. Por ejemplo, los agricultores ahora pueden monitorear la temperatura del suelo y la humedad desde sus casas.

2.1.1.4. Aplicaciones de infraestructura

La supervisión y control de las operaciones de infraestructura urbana y rural también son actividades clave en el ámbito de IoT. Entre estas pueden encontrarse la construcción de puentes, vías férreas, parques eólicos en tierra y en alta mar, etc. La infraestructura de IoT se puede utilizar para monitorear cualquier evento o cambio en las condiciones estructurales que puedan comprometer la seguridad. El IoT puede beneficiar a la industria de la construcción a la hora de ahorrar gastos, reducir el tiempo, generando jornadas de mayor calidad y aumentando la productividad. También, puede ayudar a tomar decisiones más rápidas y ahorrar dinero con el análisis de datos en tiempo real, así como también para programar actividades de reparación y mantenimiento de manera eficiente, coordinando tareas entre diferentes proveedores de servicios y personal de estas instalaciones. Los dispositivos de IoT también pueden utilizarse para controlar infraestructura crítica como puentes para proporcionar acceso de los barcos a los puertos. Los beneficios que genera el IoT aplicado en infraestructura mejora la gestión de incidentes y la coordinación de respuestas ante emergencias, la calidad de los servicios, los tiempos de actividad y la reducción de costos de todas las operaciones [6], [7].

- **Despliegues a escala metropolitana**: uno de los ejemplos de implementaciones de IoT a gran escala es en Songdo, Corea del Sur, que es la primera ciudad inteligente completamente cableada y con equipada con dispositivos inteligentes. Se está construyendo gradualmente, con aproximadamente el 70 % del distrito comercial completado a partir de junio del 2018, y se planea que gran parte de la ciudad termine en su mayoría automatizada, con poca o ninguna intervención humana. Otro ejemplo es el de la empresa New York Waterways, que conectó todas las embarcaciones de la ciudad de Nueva York con el fin de monitorearlas en tiempo real a toda hora. La red fue diseñada y desarrollada por Fluidmesh Networks, una compañía con sede en Chicago que desarrolla redes inalámbricas para aplicaciones críticas. Con la red inalámbrica establecida, NY Waterway puede tomar el control de su flota y de sus pasajeros de manera que antes no era posible, las nuevas aplicaciones pueden incluir seguridad, administración de energía y de toda la flota, señalización digital, Wi-Fi público, emisión de boletos sin la utilización de papel, etc.

- **Gestión energética**: un gran número de dispositivos que consumen energía como interruptores, tomas de corrientes, bombillas, televisores, etc., ya integran conectividad a internet, lo que les permite comunicarse con empresas de servicios públicos para equilibrar la generación de energía y el uso de la misma para optimizar y controlar su consumo. Estos dispositivos permiten el control remoto por parte de los usuarios, o la administración central a través de una interfaz basada en la nube, y habilitan funciones como el encendido y apagado remoto de sistemas de calefacción, control de hornos, cambios de condiciones en cuanto a los horarios de iluminación, etc.

- **Monitoreo ambiental**: las aplicaciones de monitoreo ambiental de IoT generalmente usan sensores para ayudar en la protección ambiental al monitorear la calidad del aire y del agua, las condiciones atmosféricas y del suelo, las alertas tempranas de terremotos y tsunamis, e incluso pueden incluir áreas como el monitoreo de hábitats de animales y el movimiento de los mismos. Los dispositivos IoT enfocados a esta área generalmente abarcan una gran área geográfica y también pueden ser dispositivos móviles. Se ha argumentado que la estandarización del IoT aplicado a tareas ambientales está generando un gran impacto positivo a nivel mundial.

2.1.1.5. Aplicaciones militares

El internet de las cosas militares (IoMT) es la aplicación de dispositivos inteligentes en el dominio militar a los efectos del reconocimiento, vigilancia, y otros objetivos relacionados con el combate. Está fuertemente influenciado por las perspectivas de futuras posibles guerras en entornos urbanos, lo que implica el uso de sensores, municiones, vehículos, robots, biométrica portátil y tecnología inteligente que sea relevante en el campo de batalla [8], [9].

- Internet de las cosas del campo de batalla: en el 2017, el Laboratorio de Investigación del Ejército de Estados Unidos (ARL), enfocado en la ciencia básica que intenta mejorar las capacidades de los soldados de sus ejércitos mediante tecnología, lanzó el "Internet of Battlefield Things Collaborative Research Alliance" (IoBT-CRA), apuntando a la colaboración entre el trabajo industrial, la universidad y los investigadores del ejército para avanzar en los fundamentos teóricos de las tecnologías de IoT y sus aplicaciones para los militares.
- Océano de las cosas: el proyecto Ocean Of Things es un programa militar comandado por DARPA (Defense Advanced Research Projects Agency) o Agencia de Proyectos de Investigación Avanzados de Defensa, creado para implementar IoT en grandes áreas oceánicas con el propósito de recopilar, monitorear y analizar datos ambientales y embarcaciones. El proyecto implica el despliegue de aproximadamente 50.000 flotadores que albergan un conjunto de sensores pasivos, que detectan y rastrean de manera autónoma a embarcaciones militares y comerciales como parte de una red basada en la nube.

2.1.1.6. Aplicaciones en el deporte

Finalmente, existe otro grupo, IoT aplicado al deporte, tema de interés y en el que se basa esta tesina. La tecnología de Internet de las Cosas, los objetos conectados y todos los datos que los acompañan están cambiando la historia en el mundo del deporte. En sus comienzos, empezaron incursionando en los ámbitos profesionales, pero ahora la revolución también está al alcance de la mano de los aficionados o

para las personas que lo practican. Se estima que más del 40 % de la Unión Europea hace ejercicio regularmente, y esa es una de las razones por la cual el IoT aplicado al deporte está creciendo a nivel exponencial [10], [11].

La tecnología y el deporte siempre han estado caminando de la mano, ya sea para mejorar los entrenamientos, reforzar la seguridad de las prácticas y la imparcialidad de los eventos, o incrementar la experiencia de entrenamiento de los deportistas. Uno de los mejores ejemplos, son los Juegos Olímpicos de Londres 2012 y sobre todo Río 2016, donde se destacó el gran uso de los dispositivos wearables, cámaras y otros dispositivos de IoT en todas las áreas de competencia. Aun así, se dice que la verdadera revolución llegará en Tokio 2020, donde se utilizará la red 5G que conectará a los cientos de millones de dispositivos que estarán en funcionamiento y las personas podrán seguir en tiempo real todos los datos de los atletas. Según anuncian los pronósticos, se espera que para el 2022 haya un volumen de 29.000 millones de dispositivos conectados, de los cuales 18.000 millones serán objetos IoT, todos comunicándose e intercambiando datos a través de una infraestructura cada vez más robusta.

Existe un gran número de dispositivos, desde Smart Watches hasta sensores incorporados en la vestimenta de los deportistas, de esta forma, pueden obtenerse datos exactos y en tiempo real de su velocidad, distancia recorrida, movimientos realizados, sprints, aceleraciones, etc. Por ejemplo, para los nadadores existen dispositivos que incluyen acelerómetros y giroscopios, permitiendo medir el tiempo, la velocidad, la aceleración y hasta realizar un análisis en 3D de los movimientos del nadador, con información sobre el desplazamiento real del agua con cada brazada. Cada vez es más frecuente la utilización de monitores que nos brindan una visión global del estado físico del deportista, a través de datos como el ritmo cardiaco, el pulso, o las calorías quemadas. Con estos análisis, los deportistas pueden "conectarse" más con sus cuerpos aumentando así su competitividad gracias al los resultados extras que pueden ver con los datos obtenidos de los dispositivos.

De este modo, los dispositivos IoT se han convertido en una red de seguridad para los deportistas y preparadores físicos, ayudando incluso a prevenir lesiones y cuidar más de su salud. Un ejemplo en cuanto a las lesiones es el equipo de fútbol Rosenborg BK de Noruega, donde consiguieron reducir las lesiones de sus jugadores en un 50 % desde que comenzaron a invertir en dispositivos de IoT con sus respectivos sensores

y su correspondiente sistema de monitoreo de los datos. Sabemos la competitividad que presenta el ámbito del deporte y este es aún más competitivo cada año, por eso la información que aporta la tecnología IoT es una ventaja adicional que puede ser incluso diferencial y determinante en deportistas y equipos de alto rendimiento [12], [13], [14].

2.1.2. Tendencias y características

La característica más significativa en IoT en la actualidad es el crecimiento exponencial de dispositivos conectados y controlados a través de internet. La gran variedad de áreas y aplicaciones para la tecnología IoT lleva a que los detalles entre estas puedan diferir entre cada dispositivo, pero hay otras características básicas que son compartidas por la mayoría. El IoT trae amplias ventajas en cuanto a la conexión del mundo físico y los sistemas informáticos, dando como resultado mejoras en la eficiencia, beneficios económicos y reducción de esfuerzos humanos. El número de dispositivos IoT aumentó un 31 % para el año 2017 con 8,4 millones de dispositivos conectados, y se calcula que habrá más de 30 mil millones de dispositivos para el final del 2020. A su vez, se estima que el valor de mercado global de IoT llegará los 7,1 billones de dólares el fin del 2020.

2.1.3. Arquitectura

La arquitectura IoT en su versión simplista consta de tres niveles: Nivel 1: dispositivos, Nivel 2: Edge Gateway y Nivel 3: la nube. Los dispositivos incluyen elementos conectados en red, como los sensores y actuadores que se encuentran en los equipos IIoT, específicamente aquellos que utilizan protocolos como Modbus, Bluetooth, Zigbee o protocolos usados para conectarse a un Edge Gateway. Edge Gateway está formado por sistemas de sensores que brindan funcionalidad, como el preprocesamiento de los datos, la seguridad de la conectividad a la nube, el uso de WebSockets, el centro de eventos, y también en algunos casos el análisis de “Fog computing” o computación en la niebla (arquitectura que utiliza edge devices para llevar a cabo una cantidad sustancial de computación, almacenamiento y comunicación a través de la red). La capa Edge Gateway también se requiere para dar una vista común

por parte de los dispositivos a las capas superiores para facilitar la administración. A nivel final incluye la aplicación en la nube creada para IIoT usando la arquitectura de microservicios, que generalmente es de naturaleza inherentemente segura usando el protocolo HTTPS/OAuth. Incluye varios sistemas gestores de bases de datos que almacenan los datos de los sensores, como base de datos orientadas a series de tiempo, por ejemplo InfluxDB (que hablaremos más adelante), o bases de datos orientadas a documentos como Cassandra o MongoDB, o las clásicas bases de datos relacionales como PostgreSQL y MySQL. El nivel de la nube en la mayoría de los sistemas de IoT basado en la nube presenta un sistema de mensajes y colas de eventos que maneja la comunicación que ocurre en todos los niveles [15].

La arquitectura de IoT debe cumplir ciertos requisitos para que esta tecnología sea viable. Por ejemplo, debe permitir que la tecnología sea distribuida (donde los objetos puedan interactuar entre ellos), debe ser escalable, eficiente y segura.

2.1.3.1. Tecnología distribuida

Una de las principales bases por la cual se fundamenta esta tecnología es poder comunicar dispositivos conectados alrededor de nuestro entorno, esto significa que la información y los datos adquiridos puedan provenir de diferentes lugares a la vez, procesada por computadoras o servidores diferentes. Todo esto significa que una gran cantidad de objetos, dispositivos y máquinas separadas físicamente se conecten entre sí mediante una red de comunicaciones, cada componente con su propio software y hardware. Es por esto que la arquitectura debe ser capaz de mostrar todos los componentes como un único sistema a los ojos de los usuarios y desarrolladores. Este concepto no es algo nuevo, lo podemos ver en los servicios en la nube como Google Drive, donde no es importante donde realmente se están guardando los datos como tampoco es de importancia desde donde los subamos, para los usuarios es todo un gran sistema al que accedemos y enviamos o recibimos datos.

Uno de los grandes problemas que nos encontramos es la estandarización de los protocolos de comunicación. Esto significa que si cada fabricante utiliza su propia tecnología, los dispositivos no podrán comunicarse entre sí. Este aspecto no puede permitirse en el mundo IoT, donde es necesaria la interacción y el intercambio de

datos entre cualquier dispositivo y de manera bidireccional. Un ejemplo de esta problemática sería imaginar que en internet no existieran protocolos, y que cada servidor web tuviera su propio protocolo y su propio lenguaje de programación, los desarrolladores tendrían que programar y especializarse en cada servidor particularmente, teniendo que crear tantas versiones de páginas web como protocolos existan, algo que está muy lejos de ser algo viable. Es por eso, que el IoT intentará aprovecharse de los estándares ya existentes y a su vez crear nuevos para que la información fluya sin obstáculos.

2.1.3.2. Arquitectura escalable

La arquitectura IoT debe ser escalable en el tiempo, esto requiere que pueda soportar cada vez más dispositivos conectados a la red, algo de suma importancia debido al crecimiento exponencial de la cantidad de dispositivos conectados a internet. A lo largo del tiempo se sumarán cada vez más objetos inteligentes, se calcula que de 50 a 100 billones de objetos alrededor del mundo y donde se podrá seguir su movimiento en tiempo real. Los seres humanos en entornos urbanos esta en promedio rodeados de 1000 a 5000 objetos rastreables. En 2015 y había 83 millones de dispositivos inteligentes en los hogares de las personas. Se espera que este número aumente a 193 millones de dispositivos para finales del 2020 [16].

Uno de los grandes problemas es identificar a cada dispositivo conectado. El protocolo existente y utilizado en la actualidad es el IPv4 que puede generar 4.294.967.296 IPs, un número inadecuado para dar una dirección a cada persona del planeta y mucho menos a cada dispositivo, teléfono, PDA, tablet, etc. A principio del año 2010 quedaban menos del 10 % de IP sin asignar, pero en el 2011 la IANA (Agencia Internacional de Asignación de Números de Internet) entregó el último bloque de direcciones disponibles (33 millones) a la organización encargada de asignar las IPs en Asia, un continente donde existe un enorme mercado en crecimiento y que no tardará en utilizarlas todas. La solución en marcha a este problema es la implementación del protocolo IPv6, que admite 2^128 IPs (340 sextillones de direcciones), cerca de 6,7 x 10^17 de direcciones por cada milímetro cuadrado de la superficie de la Tierra [17].

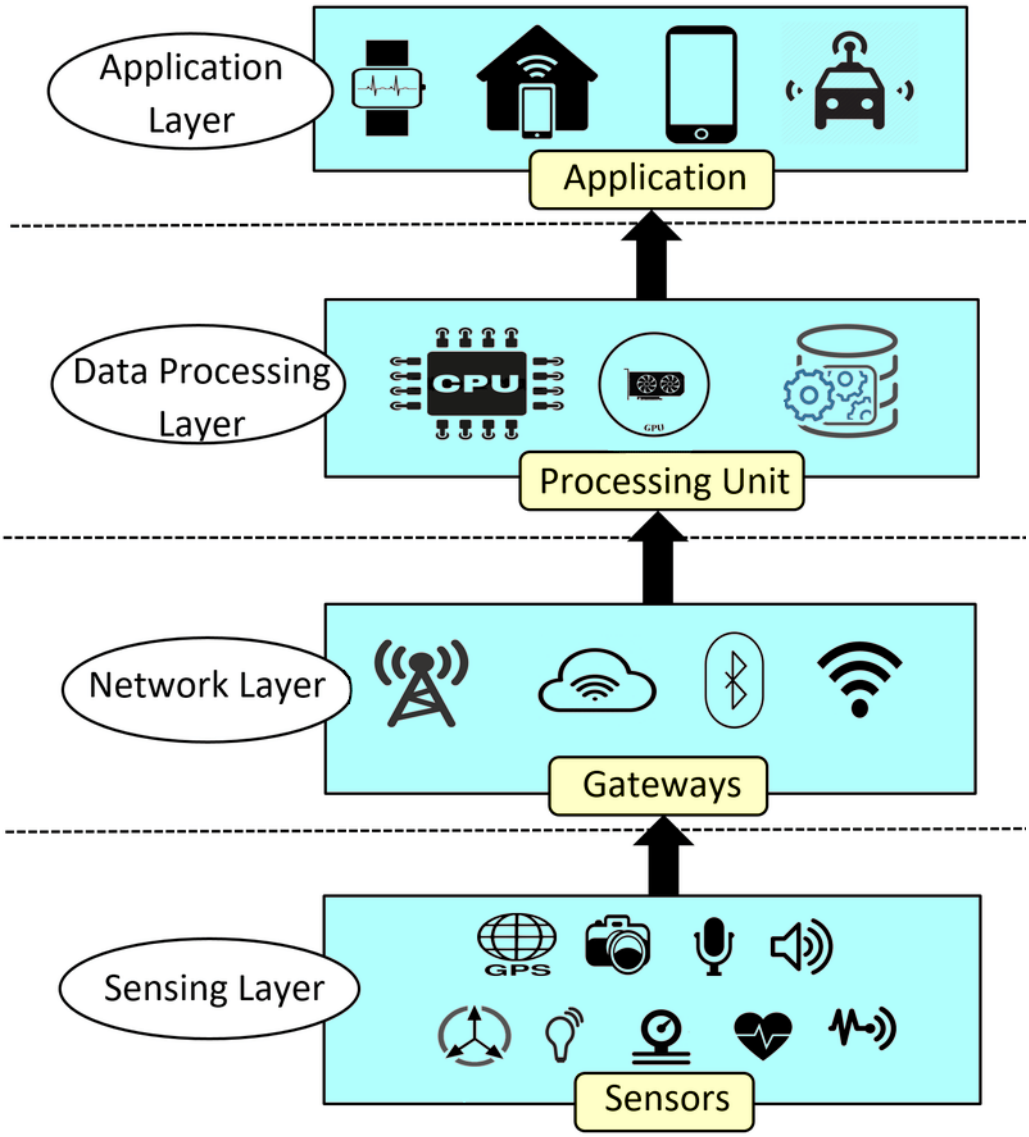


Figura 2.1: Capas de la arquitectura IoT.

2.1.4. Eficiencia energética

Si lo que necesitamos son dispositivos autónomos conectados, estos deben poder funcionar la mayor cantidad de tiempo posible sin necesidad de recargar su batería. Esto puede mejorarse de diferentes maneras, ya sea impactando en la fabricación de las baterías para que sean más duraderas, o mejorando la eficiencia de los dispositivos. Este aspecto es uno de los retos más grandes a enfrentar en lo que respecta a tecnologías IoT en la vida diaria, y es por eso que a su vez se está intentando

integrarlo con energías renovables e inagotables.

2.1.5. Complejidad

El IoT se considera como un sistema complejo debido a una gran cantidad de enlaces diferentes, interacciones entre actores autónomos y su capacidad para integrar con nuevos objetos. Como enfoque práctico, no todos los elementos en internet de las cosas se ejecutan en un espacio público global, los subsistemas a menudo se implementan para mitigar los riesgos de privacidad, control y confiabilidad, por ejemplo en el área de la domótica donde el sistema se ejecuta dentro de una casa inteligente donde los datos pueden compartirse y estar disponibles a través de una red local. La gestión y el control de una red de dispositivos de IoT es una tarea difícil con la arquitectura de redes tradicional, la red definida por software (SDN) proporciona una solución ágil y dinámica que puede hacer frente a los requisitos especiales de la diversidad de IoT y sus innovadoras aplicaciones [18].

2.1.6. Seguridad

La seguridad en el ámbito del IoT es una de las mayores preocupaciones ya que se está produciendo un rápido desarrollo sin una consideración adecuada de los profundos desafíos de seguridad involucrados, ni de los cambios regulatorios que podrían ser necesarios.

La mayoría de las inquietudes técnicas de seguridad son similares a las de los servidores convencionales, estaciones de trabajo y teléfonos inteligentes, que incluyen problemas como autenticación débil, olvidando cambiar las credenciales predeterminadas, mensajes no cifrados entre los dispositivos, inyecciones SQL y manejo insuficiente de actualizaciones de seguridad. Sin embargo, muchos dispositivos IoT tienen limitaciones operativas severas en la potencia computacional disponible. Estas restricciones hacen que no puedan usar medidas de seguridad básicas, como por ejemplo implementar firewalls o cifrados fuertes para encriptar los datos en la comunicación con los demás dispositivos.

Los dispositivos de IoT también tienen acceso a nuevas áreas de datos y a menudo pueden controlar dispositivos físicos. Se ha demostrado que los dispositivos controlados por computadora en automóviles, como frenos, motor, cerraduras, bocina, climatizador y tablero de instrumentos son vulnerables a los atacantes que tienen acceso a la red del vehículo. En algunos casos los sistemas computacionales de los vehículos están conectados a Internet, lo que les permite explotarse de forma remota, por ende los dispositivos IoT con acceso a internet mal asegurados también se pueden utilizar para atacar a otros. En 2016, un ataque distribuido de denegación de servicio (DDoS) impulsado por dispositivos de IoT que ejecutaban el malware "Mirai" derribó a un proveedor de DNS y a los principales sitios web del mundo. El Mirai Botnet había infectado aproximadamente a 65.500 dispositivos IoT en las primeras 20 horas, en los que destacaban en cámaras, routers, impresoras, y otros dispositivos conectados.

Algunas personas argumentan que es necesaria la intervención gubernamental para asegurar los dispositivos IoT y reforzar las medidas de seguridad y protocolos debido a que los incentivos del mercado resultan insuficientes. A su vez, se descubrió que debido a la naturaleza de la mayoría de las placas de desarrollo IoT, estas generan claves predecibles y débiles que hacen que sea fácil ser utilizados por distintos atacantes y por malwares, sin embargo, muchos investigadores propusieron varios enfoques de fortalecimiento para resolver el problema de la implementación débil utilizando SSH y métodos de encriptación más seguros [19], [20].

Capítulo 3

Tecnologías analizadas

3.1. Conceptos esenciales de Electrónica

Para el desarrollo de esta tesina, ha sido necesario investigar conceptos relacionados con la electrónica, debido a que estos temas no han sido abordados a lo largo de la carrera. A continuación se describirán conceptos más relevantes.

3.1.1. Placa de Desarrollo

Una placa de desarrollo es una placa con un circuito impreso y hardware integrado para facilitar la experimentación con distintos microcontroladores. Sin ellas, para utilizar un microcontrolador particular deberíamos crear un circuito con todo el hardware necesario como RAM, ROM, Buses, y todo el circuito eléctrico para poder desarrollar. Algunas incluso contienen LEDs, LCD, botones, puertos series, puertos USB, circuitos de Power Supply, pines de Input/Output para poder conectar y acceder a los demás componentes conectados. Principalmente tienen una interfaz de programación, que facilita la escritura de nuestro programa en la memoria ROM del microcontrolador [21].

3.1.2. Microcontrolador

Un microcontrolador es una pequeña computadora, un circuito integrado único que contiene uno o múltiples CPUs, con una memoria e inputs/outputs programables generalmente dedicados a una única tarea ejecutando un único programa. El programa que los microcontroladores corren está escrito en una memoria RAM ferroelectrica (no volátil), EEPROM o NOR Flash. Son extremadamente pequeños, de bajo consumo y realizados con componentes económicos (algunos llegan a costar centavos de dolar), se fabrican de manera masiva y cualquier dispositivo electrónico que conozcamos contiene uno o más de estos. Los microcontroladores suelen tener las siguientes características [22].

- CPU, puede variar desde pequeños procesadores de 4 bits hasta 64 bits.
- Serial Input/Outputs como serial ports u otras interfaces de comunicación.
- Conversores análogo-digital o viceversa.
- Capacidad para In-Circuit Programming (ISP)
- Almacenamientos volátiles y no volátiles.

3.1.3. Pinouts

Un pinout es una referencia a los pines o contactos de un dispositivo electrónico o conector. Suele ser un diagrama o una tabla que especifica la forma de verlo, generalmente queda más claro con un diagrama que indica la forma de ver el conector. Describe la función de lo que es transmitido por ese contacto. La Figura 2.2 muestra un ejemplo con un chip Wi-Fi ESP8266.

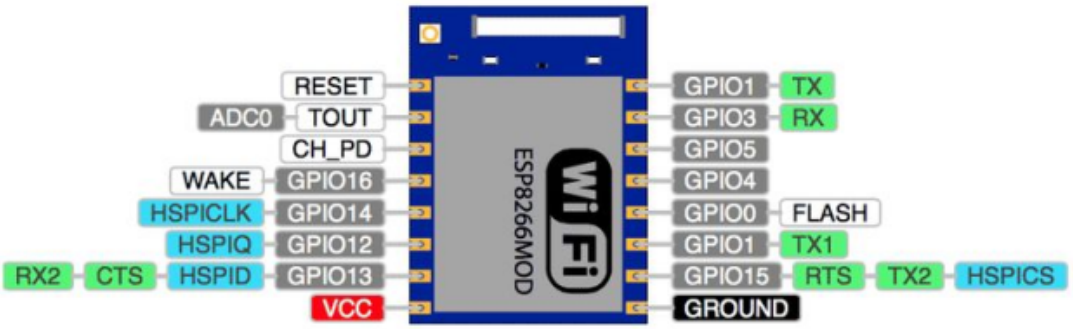


Figura 3.1: Pinout de ESP8266.

3.1.4. Serial Port y Baud Rate

Serial Port hace referencia a los dispositivos de comunicación en serie que se encuentran en las computadoras, microcontroladores y dispositivos electrónicos, son interfaces que transmiten o reciben data de 1 bit a la vez (señal binaria), en un periodo de tiempo, el número total de bits transmitidos en un segundo es llamado Baud Rates. La forma de referirse a ellos al ser conectados a la computadora es COM (Communication Port). Se pueden comunicar de manera sincrónica, uno de los dispositivos maneja un clock que controla la velocidad de comunicación, o asincrónica sin clock, pero teniendo que acordar una velocidad (Baud Rate) de antemano.

El Baud Rate mencionado previamente, se puede definir como el tiempo en el que los "altos" y los "bajos" (1 y 0 binarios) de una muestra son decodificados de una determinada señal. El estándar utilizado usualmente es 9600. Otros Baud Rates "estándar" son 1200, 2400, 4800, 19200, 38400, 57600, y 115200. Cuanto más alto el Baud Rate, más rápido se envían/reciben los datos, pero hay límites en la velocidad. Usualmente no se ven velocidades superando los 115200, es demasiado rápido para la mayoría de los controladores, pasando valores muy altos se empiezan a observar errores en los datos del receptor, ya que los relojes y el muestreo no pueden con esta velocidad. Si dos dispositivos no están coordinados a la misma velocidad, lo que se va a recibir son datos basura. [23]

3.1.5. Protocolos de Comunicación en Serie

Para poder comunicarse entre dispositivos, se tiene que establecer un protocolo, que defina el formato y la forma de transmitir los datos. Los protocolos de comunicación en serie más conocidos y que encontramos en casi todas las placas de desarrollos y sensores que se utilizan en la electrónica son USB, UART, SPI e I2C. La mayoría de estos protocolos de comunicación utilizan el modelo maestro/esclavo, en el cual tenemos un dispositivo maestro que se conecta y administra la comunicación con el esclavo o múltiples esclavos, es decir, controla la sincronización por clock y la forma de enviar los datos.

3.1.5.1. USB (Universal Serial Bus)

Universal Serial Bus o USB es uno de los protocolos más conocidos, casi todos los dispositivos que utilizamos, como celular, tablet, PC, consolas de videojuegos, etc, se conectan utilizando este protocolo. Como su nombre lo dice, es un protocolo de comunicación en serie y universal, fue diseñado con el objetivo de estandarizar la forma de conectar periféricos a las computadoras. Existen múltiples versiones del mismo, la más actual es la 3.0, en cada versión nueva se desarrollaron mejoras de velocidad llegando a los 20Gbits/s en la versión 3.2 y aumento de la cantidad de energía que provee al dispositivo (utilizado usualmente para cargar baterías). Por otro lado, el mismo ha sufrido modificaciones en su tipo de conector, respecto a su cantidad de pines y formato, actualmente su forma más común es la Tipo C, la cual es reversible (se puede conectar sin importar si el cable está boca arriba o abajo). No es utilizado como protocolo de comunicación chip a chip, o chip a periféricos en circuitos integrados.

3.1.5.2. UART (Universal Asynchronous Receiver and Transmitter)

Protocolo de comunicación asincrónica en serie, aunque también tiene su versión capaz de trabajar en modo de comunicación sincrónica y asincrónica (USART). Utilizado principalmente en comunicación de circuitos integrados. Para el modo asincrónico usa solamente dos cables, Rx (recepción) y Tx (transmisión). Como en

este modo no se utiliza un clock, los dos dispositivos conectados necesitan usar su clock interno. El Baud Rate define a la velocidad para que los dos dispositivos estén sincronizados. La mayor limitación de este dispositivo es que solo dos dispositivos pueden estar conectados comunicándose a la vez. El pin Tx de uno de ellos va conectado al Rx del otro, similarmente el Tx de este último va conectado al Rx del primer dispositivo. Suele ser el protocolo más simple de utilizar.

3.1.5.3. I2C (Inter-Integrated Circuit)

Es un protocolo en serie sincrónico que utiliza dos cables para todo su proceso por lo cual también es conocido como Two Wire Interface (TWI) protocol, utilizado usualmente en conexiones de baja velocidad a periféricos desde un microcontrolador o procesador. Estos dos cables son el SDA (Serial Data) y SCL (Serial Clock), siendo el SDA el único que se utiliza para transmitir datos mientras el SCL mantiene la sincronización con un Clock que lo provee el dispositivo maestro activo. Soporta múltiples dispositivos maestro y esclavo, cada uno de los esclavos tiene una dirección única para ser identificado. Por la naturaleza de este protocolo, utilizado para conexiones microcontrolador a sensor, las velocidades van desde los 100kbit/s a los 5Mbit/s. Es utilizado principalmente en sistemas embebidos.

3.1.5.4. SPI (Serial Peripheral Interface)

Protocolo de comunicación sincrónica que consiste en dos líneas de datos, una línea de clock y una línea de selección de esclavo. Las dos líneas de datos que posee le permiten transmitir y recibir al mismo tiempo, lo cual aumenta la velocidad final. Al igual que I2C, fue también diseñado especialmente para conexiones de microcontroladores. En SPI, solo puede haber un dispositivo maestro y múltiples esclavos, pero uno solo responderá a la llamada del maestro en un determinado momento. Toda la lógica de comunicación la realiza el dispositivo maestro, y sin su orden los esclavos no pueden enviar datos. Las líneas que este protocolo posee son:

- **MOSI**: Master Out Slave In (Envía datos a los esclavos, Tx para maestro, Rx para los esclavos).

- **MISO**: Master In Slave Out (Los esclavos responden al maestro por esta línea, Rx para maestro, Tx para esclavos).
- **SCK**: Serial Clock (clock provisto por el maestro).
- **SS, Slave Select** (utilizado para seleccionar el esclavo con el cual se comunicará)

3.2. Hardware: sensores y componentes

Un sensor se define como un dispositivo, que recibe y responde a un estímulo físico del medio en el que se encuentra. El estímulo al que nos referimos es una cantidad, propiedad o condición que es "sentida" y convertida a la señal eléctrica, esto es realizado mediante el mecanismo de transducción, el cual convierte las señales físicas a señales eléctricas (convierte un tipo de energía a otro). El objetivo del sensor es responder al estímulo y convertirlo en una señal eléctrica compatible con un circuito electrónico, no funciona por su cuenta; por sí un sensor no hace más que tomar muestras, las cuales debemos procesar y aplicarles algún tipo de lógica para generar respuestas en base a los datos. Es decir, tiene que ser integrado en un sistema. Muchas veces forman parte grandes sistemas de control que incluyen mecanismos de retroalimentación [24].

Existen dos tipos principales de sensores:

- **Pasivo**: un sensor pasivo no necesita energía adicional, su salida es provista por la señal medida sin ningún voltaje externo. La energía del estímulo es convertida por el sensor en la señal de salida.
- **Activo**: requiere una fuente de poder externa que provee casi todo el poder de salida de la señal, la cual es llamada señal de excitación.

Por otro lado, otra clasificación existente para los sensores es si es analógico o digital:

- **Sensor Digital**: la señal producida por el sensor es binaria, es decir una señal discreta que se puede representar como ondas cuadradas (pulsos), es

decir bajos y altos eléctricos, información binaria. Sus parámetros son, altura del pulso, duración y frecuencia de repetición (pulsos por segundo).

- **Sensor Analógico**: la señal producida por el sensor es continua y proporcional a lo medido, es decir una variación continua en el tiempo. No hay interrupciones en ella. A diferencia de una señal digital que tiene un valor discreto en cada punto de muestra en el tiempo, una señal análoga tiene fluctuaciones constantes.

Cuando se trabaja con sensores, también se debe tener en cuenta que existen varios aspectos que pueden alterar los valores cuantificados por estos, principalmente:

- **Calibración**: las calibraciones son procedimientos que se realizan cuando debemos ajustar la tolerancia del sensor. Por ejemplo si debemos medir una temperatura con una tolerancia de $\pm 0.5 \circ C$ y nuestro sensor nos proporciona $\pm 1 \circ C$ debemos realizar un ajuste, llamado calibración.
- **Repetición**: se refiere a la capacidad del sensor de representar el mismo valor bajo la misma condición de estímulo. Factores externos al sensor generan este error en la medición, las cuales pueden ser temperatura, estática, degradación de los materiales del sensor, etc.
- **Saturación**: los sensores tienen límites para operar. A partir de un determinado valor que supera la capacidad del sensor, vamos a obtener datos incorrectos o no deseados.
- **Banda muerta**: se refiere al rango en el que en sensor insensible a la señal de entrada, por ejemplo un estímulo demasiado pequeño que el sensor no es capaz de detectar entra en la zona muerta.
- **Resolución**: es la medida del menor incremento o disminución en el estímulo que puede registrar.

3.2.1. Filtros de Señales

Cuando hablamos de un filtro de señal nos referimos a un dispositivo o proceso para separar y/o suprimir un grupo de señales de un conjunto de las mismas. La

separación de señal se necesita cuando esta ha sido contaminada con interferencia, ruido u otras señales.

La restauración de señal es usada cuando una señal ha sido distorsionada de alguna manera. Un ejemplo simple para entender, sería una grabación de audio de voz realizada con mal equipamiento, donde se escuchen sonidos de fondo que queremos eliminar para escuchar con mayor claridad la voz. Otro ejemplo de filtro es cuando se quita el blur (difuminado) a una foto tomada con una cámara fuera de foco.

En otras palabras, los filtros son diseñados para dejar pasar o amplificar las señales de entrada de ciertas señales de frecuencias (de tipo passband) y bloquear o atenuar ciertos rangos de frecuencias o bandas de frecuencias (tipo stopband). Por lo tanto, son extremadamente usados en la electrónica para manipular señales captadas por los sensores. Algunos de los más importantes y básicos son:

- **Low-pass Filter**: todas las señales bajas pasan, elimina o atenúa las señales altas.
- **High-pass Filter**: opuesto al Low-Pass Filter, pasan las señales altas, atenúa o elimina las bajas.
- **Band-pass Filter**: permite el paso de una banda de frecuencias y elimina todas las demás, utiliza un rango de valores a filtrar.

El filtrado se puede realizar con dispositivos electrónicos o por software, es decir con funciones matemáticas en algoritmos. Por lo tanto, puede construirse filtros más complejos en base a algoritmos, como por ejemplo los filtros de Mahony y Madgwick que más adelante se explicaremos como son utilizados por software para procesar las señales tomadas de nuestros sensores [25].

3.2.2. GPS

Existen múltiples sistemas de posicionamiento global en el mundo. El más conocido de todos es el Global Positioning System (GPS) de los Estados Unidos.

En la navegación por satélites o satnav (forma corta de satellite navigation) se utilizan satélites que proveen ubicación geo espacial, le permite a receptores de señal electrónicos determinar su ubicación de manera precisa usando señales de tiempo transmitidas por la radio de los satélites. Los satélites transmiten 3 piezas de información fundamentales, las cuales son su número de satélite, la posición en el espacio y el tiempo en el que la información es enviada. En adición a esto, los GPS pueden enviar también data de su velocidad y dirección de viaje en la atmósfera. Estas señales que envían los satélites son tomadas por un receptor. Cabe destacar que funcionan independientemente de las redes telefónicas o de internet.

Un sistema de navegación de satélites con alcance global se le llama Global Navigation Satellite System (GNSS). Debido a los sistemas GNSS proveen posicionamiento tridimensional en tiempo real 24 horas al día, 7 días a la semana, alrededor de todo el mundo, con una alta precisión, se utilizan de manera masiva en numerosas aplicaciones, incluyendo monitoreo, mapeo y recolección de data GIS (geographic information system).

Debido a que el GNSS más común de todos es el GPS, en la actualidad la mayoría de la población mundial se refiere a los sistemas de navegación global con el nombre GPS de manera genérica. En este documento nos referiremos a los sistemas de posicionamiento global como GPS, no necesariamente haciendo referencia al sistema estadounidense a menos que se aclare lo contrario.

Estos sistemas se dividen en tres segmentos. El segmento de *espacio*, es el que consiste en los satélites dispersados en las diferentes órbitas. El segmento de *usuarios* consiste en los receptores, que puede incluso ser pequeños dispositivos como nuestros Smartphones. Cualquier tipo de receptores GPS, procesadores y antenas utilizadas para posicionamiento y sincronización utilizados por la comunidad y las fuerzas armadas. Por último, el segmento de *control* consiste en múltiples estaciones terrestres, ubicadas alrededor del mundo, que se encargan de verificar que los satélites funcionan de manera correcta [26].

3.2.2.1. Satnav Regionales

Varios países y regiones terrestres tienen sus propios satélites que conforman un satnav. De estos satnav, GPS y GLONASS se consideran Fully Operational.

- Estados Unidos utiliza **Global Positioning System (GPS)**, consiste en 32 satélites de Órbita circular intermedia (MEO, acronimo de Medium Earth Orbit, a una altitud de aproximadamente 20mil kilómetros), ubicados en 6 planos orbitales distintos. El número de estos satélites va variando a medida de que las unidades más viejas van siendo retiradas y reemplazadas por nuevas. Esta operacional desde 1978, pero no se hizo disponible al público hasta 1994 (previamente sólo era utilizado por el ejército estadounidense) [27] [28].
- Rusia utiliza el **Global NAvigation Satellite System (GLONASS)**, principalmente usado por las fuerzas Rusas de Defensa Aeroespacial, consiste en 24 satélites en Órbita Circular Intermedia con una altitud de aproximadamente 23mil kilómetros [29].
- **BeiDou Navigation Satellite System** es el satnav de China, se le suele conocer como BeiDou simplemente
- **Galileo**, el GNSS de la Unión Europea, consiste en una constelación de 30 satélites, que actualmente tiene 22 funcionales (operacionales y contribuyendo actualmente a los servicios de posicionamiento), 2 de testeo, y el resto no funcionales/retirados. Su objetivo es proveer ubicación en caso de que GPS y GLONASS dejen den de proveer servicio o sus capacidades sean limitadas para uso público [30].

La forma de conseguir cobertura global con un sistema de satélites es tener una constelación de satélites de entre 18-30 satélites desplegados en los distintos Planos Orbitales. Estos tardan 12 horas en orbitar la tierra, cada uno está equipado con un reloj atómico (tipo de reloj que para alimentar su contador utiliza una frecuencia de resonancia atómica normal), estos relojes son los más precisos en el mundo, teniendo un retraso de 1 segundo cada 30 millones de años. La unidad terrestre recibe la señal, que incluso viajando a la velocidad de la luz, tarda un tiempo considerable en llegar a la tierra [31],[32], [33].

3.2.2.2. Trilateración

La técnica utilizada para el cálculo de una posición se llama trilateración, generalmente se tiene un concepto erróneo de los mismos pensando que la forma de calcular la ubicación es la Triangulación (que calcula ángulos pero esto no es así). La trilateración mide distancias.

Es un método matemático para calcular la ubicación de un objeto en una superficie. Se puede decir que esta técnica es una versión de la triangulación que no utiliza, como previamente dijimos, los ángulos en sus cálculos. Estando en B, queremos conocer su posición relativa a los puntos de referencia P1, P2, y P3 en un plano bidimensional. Al medir r1 se reduce nuestra posición a una circunferencia. A continuación, midiendo r2, la reducimos a dos puntos, A y B. Una tercera medición, r3, nos devuelve nuestras coordenadas en B. Una cuarta medición también puede hacerse para reducir y estimar el error.

3.2.2.3. Detectando la ubicación

Explicado de manera simple sin entrar en extremo detalle de la matemática subyacente, la diferencia de tiempo en el que la señal es enviada y el tiempo en el que es recibida, multiplicada por la velocidad de la luz, le permite al receptor calcular la distancia al satélite. Todos los satélites realizan un broadcast, es decir envían su señal de manera que cualquier receptor de ese tipo de señale la pueda recibir con un tiempo específico y distancia. Suponiendo que estamos posicionados con un receptor de GPS, este puede ser nuestro celular en una plaza, cuando un satélite envía su señal que eventualmente llega a nuestro receptor, no existe forma de saber nuestro ángulo en relación al satélite, pero si tenemos forma de calcular la distancia. Es por eso que que la distancia forma un círculo igual en todas las direcciones, es imposible saber exactamente en qué punto estamos pero si la distancia a la cual estamos de ese punto. De esta forma se genera un radio, y nuestra posición de GPS podría ser cualquiera en ese radio específico. Con esta única señal no podemos obtener ningún dato preciso, nos falta más información para saber dónde estamos exactamente.

La segunda señal se transmite de la misma manera que la del primer satélite hasta

que llega a nuestro receptor de gps, de esta forma nuestra ubicación también puede ser cualquiera posición dentro del radio de la misma. Pero esta vez, tenemos dos distancias obtenidas de dos satélites distintos, de esta forma si imaginamos que los dos radios se intersectan/superponen, nosotros estaríamos ubicados en el área dentro de la intersección de los mismos. La Figura 2.3 ilustra esta situación. Estaríamos necesitando un tercer satélite para obtener nuestra verdadera ubicación en la superficie terrestre, utilizando la técnica de trilateración.

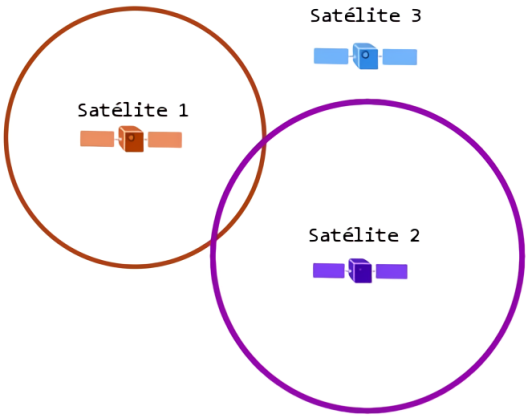


Figura 3.2: Intersección de los radios de los dos satélites.

En un escenario de mundo real y no en papel que representa dos dimensiones, estos círculos se transforman en esferas, y por lo tanto necesitamos un cuarto satélite para tener la ubicación precisa. Es decir, en caso de ubicación bidimensional necesitamos tres satélites, y en caso de ubicación tridimensional se utilizan cuatro, debido a la necesidad de saber la elevación. Cabe aclarar que podemos utilizar aún más satélites para tener una posición más precisa, teniendo receptores que llegan a conectarse a más de 14 satélites en simultáneo para mayor precisión. Aun así, esto no es lo único que define la precisión de la ubicación obtenida, existen múltiples factores como por ejemplo la atmósfera, los conceptos de HDOP, PDOP, GDOP que serán explicados más adelante [34].

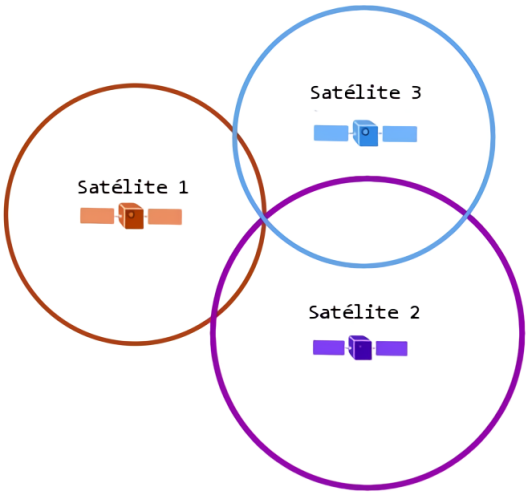


Figura 3.3: Tres satélites suponiendo un plano bidimensional.

3.2.2.4. NMEA, el protocolo de comunicación

NMEA, el acrónimo de National Marine Electronics Association, es una organización estadounidense dedicada la mejora y organización de los estándares de comunicación de dispositivos marinos. Debemos aclarar que dispositivos marinos hace referencia a los dispositivos electrónicos utilizados para geolocalización, pudiendo ser sonares, radares marinos, GNSS, anemómetros, girocompases, pilotos automáticos, pero no significa que no puedan ser utilizados en tierra firme. La NMEA creó una interfaz estándar que define la forma de comunicarse de estos dispositivos electrónicos. El estándar le permite enviar información a las computadoras y distinto tipo de equipamiento apto para recibirla.

NMEA 0183, es el estándar utilizado por los receptores GPS. Esta data incluye la posición, velocidad y tiempo (PTV) calculada por los receptores. La idea de las sentencias NMEA es enviar una línea de datos llamada sentencias totalmente autónoma e independiente del resto de las sentencias, fácilmente identificables (por ejemplo las de GPS tienen el prefijo GP). A su vez NMEA permite a los fabricantes de hardware definir sus sentencias propietarias para el propósito que crean necesario (por ejemplo Garming y sus sentencias PGRM) [35].

Las sentencias tienen un prefijo que indica el tipo de dispositivo que usa ese tipo de sentencia, el cual es precedido por tres letras que indican el contenido de las

sentencias. Todas las sentencias empiezan con un '$' y terminan con un retorno de carro (CR, Carriage Return), más conocido como lo que se produce al presionar la tecla Enter. No puede ser más largo que 80 caracteres incluyendo el CR. Los ítems en la sentencia están separados por una coma, y la data en si es solamente texto ASCII que puede estar extendida en múltiples sentencias. Al final de cada sentencia hay un Checksum que puede o no ser utilizado para ver si los datos están bien. El campo de Checksum consiste en un asterisco '*' y dos dígitos hexadecimales representando un OR exclusivo de 8 bits entre todos los caracteres, pero no incluyendo el '$' y el '*'.El Checksum es requerido en algunas sentencias. Existen varias versiones del estándar así que puede ser necesario especificar cuál se está usando para ser compatible [36],[37].

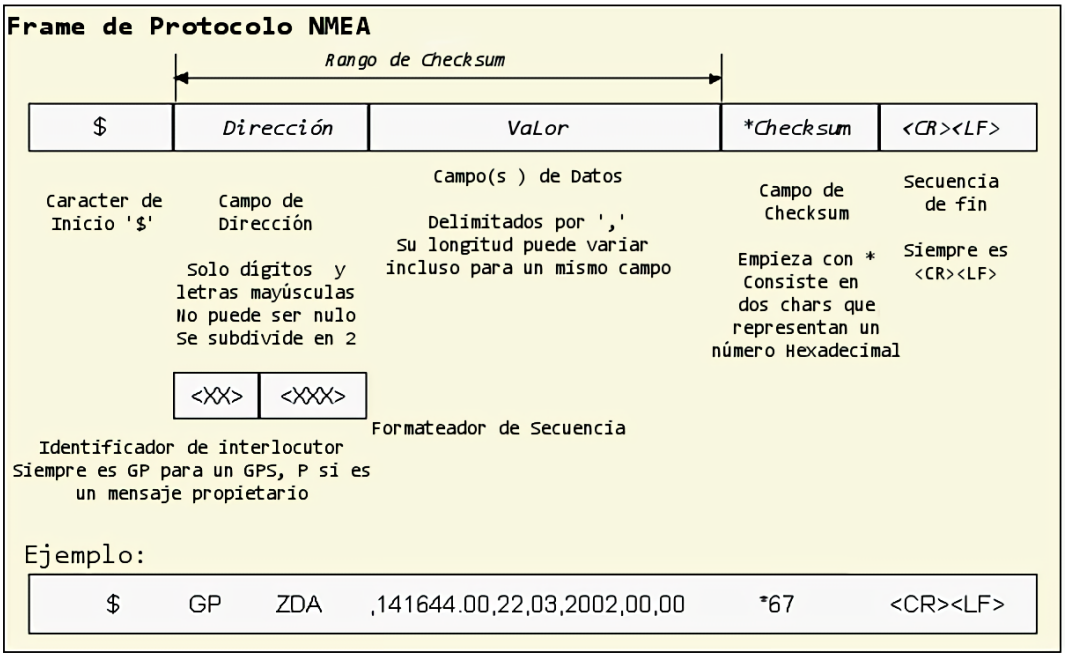


Figura 3.4: Ejemplo de Frame del protocolo NMEA.

Cuando hablamos de estas sentencias, un FIX se refiere a la información de localización que el sistema GNSS nos da para un punto específico. De ahora en adelante utilizaremos el término FIX para referirnos a esta data.

NMEA abarca una amplia cantidad de sentencias, cada una de ellas teniendo múltiples datos. Las 3 sentencias más importantes son:

- **GGA**: data esencial del Fix que provee los datos de ubicación 3D y precisión.
- **RMC**: NMEA tiene su propia versión de los datos esenciales de GPS PVT (posición, velocidad y tiempo), estos son llamados RMC, Recommended Minimun.
- **GSA**: GPS DOP y satélites activos. Esta sentencia provee detalles de la naturaleza del FIx. Incluye el número de satélites que están siendo usados en el momento y el DOP (Dilution Of Precision), que es una indicación del efecto de la geometría del satélite en la precisión del Fix.

Descripción de los campos de una sentencia GGA:

$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47

Campo	Descripción
GGA	Global Positioning System Fix Data
123519	Fix tomado a las 12:35:19 UTC
I + D	Fix tomado a las 12:35:19 UTC según el reloj atómico.
4807.038,N	Latitud 48 grados 07.038' Norte
01131.000,E	Longitud 11 grados 31.000' Este
1	Calidad de Fix. Valor con distintos significados que va del 0 al 8.
08	Número de satélites que están siendo trackeados.
0.9	Dilución Horizontal de la posición (propagación de error).
545.5, M	Altitud, en metros, sobre el nivel del mar.
46.9 M	Altura respecto al geoid (mean sea level), en base al modelo World Geodetic System (WGS) WGS84.
(Vacío)	En este caso vacío, es el tiempo en segundos desde el ultimo update DGPS si el valor de la calidad de Fix es 2.
(Vacío)	ID de la estación DGPS.
***47**	Suma de comprobación, siempre empieza con *.

3.2.3. Acelerómetro

Los acelerómetros son pequeños dispositivos electromecánicos que funcionan percibiendo la aceleración de la gravedad en los distintos ejes del dispositivo. Miden en

unidades de metros sobre segundo cuadrado (m/s2) o en fuerza G (g), y la mayoría de los dispositivos tienen un rango seleccionable de fuerza que pueden medir, las cuales suelen variar entre ±1g y ±250g (cuanto más chico el rango, más sensible a pequeñas alteraciones será el acelerómetro). La aceleración se puede definir en simples palabras como la la magnitud física que mide la tasa de variación de la velocidad respecto del tiempo. Los dispositivos MEMS, Micro Electro-Mechanical Systems son pequeños circuitos integrados que además de ser electrónicos, están conectados a microscópicas piezas mecánicas utilizadas para realizar las mediciones. Esto se debe a que la gravedad es un fenómeno mecánico y no eléctrico, aunque esto no significa que deban ser estrictamente mecánicos. Existen tres tipos de acelerómetros principales:

- **Acelerómetro Capacitivo MEMS**: contienen placas capacitivas, algunas de ellas son fijas, otras están conectadas a resortes que se mueven internamente cuando la fuerza de aceleración actúa sobre el sensor. Cuando las placas se mueven la capacitancia entre ellas cambia, y con esto se puede determinar la aceleración.
- **Acelerómetro Piezorresistivo**: tiene una pequeña estructura compuesta de un chip de silicio. En lugar de sentir los cambios de capacitancia en la masa sísmica (como el dispositivo capacitivo), este produce cambios de resistencia en extensómetros que son parte del acelerómetro.
- **Acelerómetros Piezoeléctricos**: tienen una pequeña estructura de cristal que emite una carga eléctrica cuando es colocado bajo estrés mecánico (aceleración, vibración, shock mecánico) en este caso aceleración.

En el caso de este desarrollo se utilizó un acelerómetro capacitivo los cuales calculan la aceleración midiendo la fuerza de la gravedad aplicada sobre sus ejes. En este caso, si una masa en movimiento altera la distancia entre dos placas de metal, medir el cambio en su capacitancia da una medida de la fuerza que está actuando [38].

Se dice que son dispositivos que miden la "aceleración correcta" (Proper Acceleration), que es la aceleración que experimenta un objeto relativo a la caída libre. Si nuestro acelerómetro cayera infinitamente hacia el suelo, las mediciones en todos los ejes darían 0. Para un mejor entendimiento, un acelerómetro en reposo posicionado

de manera relativa a la superficie de la tierra va a indicar una medición de 1G, lo que es igual a $9,8m/s$, el valor de la fuerza de gravedad en la tierra, aunque puede variar un poco según la elevación. Esto algo bastante contra-intuitivo, ya que la superficie de la Tierra ejerce una fuerza hacia arriba (recordemos que miden la aceleración de la gravedad). Por lo tanto, los valores de los ejes serian $X = 0; Y = 0; Z = 1$. Nuestro problema surge cuando esa aceleración ejercida sobre el eje Z se distribuye sobre los demás, por ejemplo cuando el dispositivo cambia su posición en el espacio, ya sea con una pequeña inclinación o colocándolo en posición vertical. Si no se realiza algún tipo acción con respecto a esto cuando intentamos medir el vector de aceleración lineal, se tendrá como resultado una constante aceleración por parte del dispositivo. La forma de "eliminar" los efectos de la gravedad de nuestro cálculo de *Aceleración Lineal* es utilizando el giroscopio, acelerómetro y magnetómetro del MPU para determinar la rotación de nuestra placa y poder sustraer la fuerza gravitacional ejercida sobre cada eje [39].

3.2.3.1. Aceleración Lineal

La aceleración lineal o tangencial se refiere a los cambios de magnitud de velocidad con respecto al tiempo sin tener en cuenta su dirección, y sin tener en cuenta la fuerza de gravedad. Esta va a ser la aceleración que usaremos para medir lo que necesitamos en nuestro dispositivo, y para poder conseguirla es necesario la utilización de un filtro. Para calcularla tomarán un papel importante los cuaterniones, estos son vectores que nos permitirán saber la orientación de un objeto en un momento dado con respecto al eje de la Tierra. La aceleración lineal se obtiene utilizando los datos del magnetómetro y del giroscopio, con estos datos se puede conseguir el cuaternión que representa la orientación del dispositivo. La salida de cualquier acelerómetro obedece a la siguiente relación:

$$AceleraciónTotal = Gravedad + AceleraciónLineal \text{ [40]}$$

3.2.4. Giroscopio

El giroscopio es un dispositivo, originalmente mecánico, utilizado para medir la orientación y velocidad angular, o también para mantener el movimiento de rotación. La velocidad angular es la velocidad a la que gira el dispositivo alrededor de un eje especificado en un sistema de coordenadas local definido por el dispositivo (ejes X,Y,Z en base al centro relativo del dispositivo). Los giroscopios utilizados actualmente son dispositivos electrónicos MEMS, pequeños y baratos al igual que los acelerómetros como explicamos previamente, aunque estos no tienen la capacidad de mantener el movimiento de rotación. Estos no necesitan un punto fijo de referencia para medir la rotación (como por ejemplo un tacómetro o un potenciómetro) [41].

Existen otros tipos también como los giroscopios de fibra óptica o los cuánticos (siendo estos los más exactos). La unidad de la velocidad angular es medida en grados por segundo (°/s) o revoluciones por segundo (RPS). La velocidad angular es simplemente una medida de la velocidad de rotación. Los giroscopios son usados usualmente para determinar orientación, como por ejemplo la de un celular o en un sistema autónomo de navegación, para la corrección de movimiento al grabar videos con una cámara de video, o balancear un robot, ya que puede medir la rotación respecto a la posición correcta, y enviar los datos necesarios de corrección, tanto como en el caso del robot o la cámara de video.

3.2.4.1. Funcionamiento de un giroscopio MEMS

El sensor MEMS dentro de un giroscopio es muy pequeño (entre 1 a 100 micrómetros, el tamaño de un cabello humano). Cuando se hace girar el giroscopio, una pequeña masa de resonancia se desplaza con los cambios de velocidad angular. Entrando en detalle, los giroscopios MEMS contienen elementos que vibran para medir el efecto Coriolis. Este movimiento se convierte en señales eléctricas de muy bajas corrientes que se pueden amplificar para ser leídas por un microcontrolador.

En nuestro caso, utilizaremos los datos provistos por el giroscopio para conocer la orientación de nuestro dispositivo, como explicamos previamente no es posible que

nuestro IMU este sobre una superficie recta, estará constantemente en movimiento, cambiando orientación, por lo tanto lo necesitamos para poder sustraer correctamente la fuerza de gravedad y encontrar la aceleración lineal [42].

Una importante aclaración respecto al giroscopio es que mientras el acelerómetro y el magnetómetro miden la aceleración y ángulo relativo a la Tierra, el giroscopio mide la velocidad angular relativa a su cuerpo.

3.2.5. Magnetómetro

Son instrumentos de medición que permiten medir la fuerza, y en algunos casos la dirección del campo magnético al que están sometidos. Miden el campo magnético terrestre y detectan anomalías magnéticas en el mismo, pueden incluso usarse como detector de metales (estos alteran el campo magnético cercano). Por eso se debe tener en cuenta que la medición que realiza es este es la suma del campo magnético terrestre y el campo magnético debido a la presencia de material magnético en las cercanías. Según lo que miden, podemos clasificar a los magnetómetros en dos tipos, escalares y vectoriales. Los magnetómetros escalares miden la fuerza total del campo magnético al cual están sujetos, los magnetómetros vectoriales, tienen la capacidad de medir el componente del campo magnético en una dirección particular, relativa a la orientación espacial del dispositivo.

Los magnetómetros que se suelen utilizar en robots, celulares y proyectos IOT son magnetómetros de 3 ejes, estos pueden detectar el magnetismo terrestre en los 3 ejes X, Y, Z.

3.2.6. Hardware y Definición de los Módulos

Cuando se inició con el desarrollo de tesina, se dispuso de un primer prototipo, construido por un estudiante durante su práctica profesional supervisada [43] compuesto por una protoboard con un IMU y GPS. Si bien las primeras pruebas se hicieron con este hardware, se decidió estudiar sobre electrónica y analizar otras componentes a fin de mejorar la performance del mismo.

- **Tamaño**: los dispositivos deben ser lo más pequeño posibles sin afectar el resultado de las mediciones de los mismos ya que el prototipo del dispositivo que integra los sensores estará en el cuerpo de un jugador en movimiento. El dispositivo debe ser liviano, compacto y que no genere molestia al jugador. Como a que estamos hablando de sensores que en la actualidad se usan de manera masiva en proyectos de IoT que necesitan cumplir esta característica, e incluso parte de los componentes corresponden a los que encontraríamos en cualquier teléfono smartphone contemporáneo, esto no generó problemas.
- **Durabilidad**: al estar sometidos a deportes de contacto como fútbol o rugby, los sensores deben soportar posibles caídas y golpes. Esta característica se provee con el encapsulado correspondiente de los sensores y su diseño final.
- **Precisión**: se busca la mayor precisión posible teniendo en cuenta que el tamaño no puede representar un impedimento y el precio de los sensores, ya que algunos de los mismos en rango profesional suelen ser excesivamente caros, por ejemplos, acelerómetros de grado industrial que se utilizan para mediciones en drones o maquinaria profesional pueden rondar los 100mil dólares.

3.2.6.1. Ublox-NEO-M8N

Este módulo GNSS fue seleccionado debido a que es utiliza recepción concurrente de hasta tres sistemas GNSS, siendo GPS/Galileo principalmente en conjunto con BeiDou o GLONASS, reconociendo múltiples constelaciones en simultáneo, aportando gran compatibilidad y seguridad en muchas partes del mundo. Además de ello, provee soporte de GNSS Augmentation, un método para mejorar atributos de los sistemas de navegación como precisión, confiabilidad y disponibilidad a través de la integración de información externa en el proceso de cálculo. Para la aumentación soporta QZSS, GAGAN, IMES, WAAS, EGNOS, entre los más importantes. Por el lado de la conectividad con el resto del hardware, el módulo posee numerosas interfaces de comunicación (SPI, UART, USB, I2C).

En el apartado de consumo eléctrico se puede decir que es capaz de trabajar en un modo de bajo consumo, esencial en dispositivos que se alimentan con baterías y no están conectados a la línea de alimentación constantemente, aunque su configuración de consumo de energía puede ser cambiada (pudiendo aumentar el mismo para

mejorar precisión). El mismo necesita como entrada un voltaje en el rango de 1,65 V a 3,6 V los cuales son valores normales en el desarrollo de dispositivos IoT. El consumo del módulo es de 21mA a 3.0V en modo Continuo y 5.3mA a 3.0V en modo bajo consumo (extremadamente bajo, con un Polling Rate de 1 Mhz).

Figura 3.5: Módulo Ublox NEO-M8N con su antena conectada.

El módulo posee optimización de radiofrecuencia, la posibilidad del cambio de la antena utilizada y un filtro SAW (Surface Acoustic Wave) para inmunidad de interferencia de señales. Soporta el protocolo NMEA, UBX Binary (propietario de U-Blox) y RTCM (para DGPS). Teniendo en cuenta que es un dispositivo de bajo costo, nos provee con una precisión extremadamente alta para su valor, siendo de de 2,5 metros aproximadamente (Posición Horizontal) utilizando GPS, de 4 metros en caso de GLONASS exclusivamente, y 3 metros en caso de utilizar BeiDou solamente [37].

3.2.6.2. MPU9250

El acelerómetro utilizado es el MPU-9250 IMU (Inertial Measurement Unit, en español Unidad de Medición Inercial) de 9 ejes (también conocido como 9DoF, Degrees of Freedom). Este es un SiP (System in Package) que combina dos chips distintos, el MPU-6500 que contiene un giroscopio de 3 ejes, un acelerómetro de 3 ejes y el magnetómetro de 3 ejes AK8963. Es por eso que al producto final se

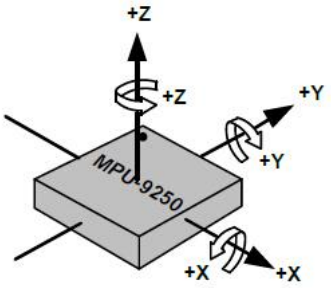


Figura 3.6: Ejes del MPU9250.

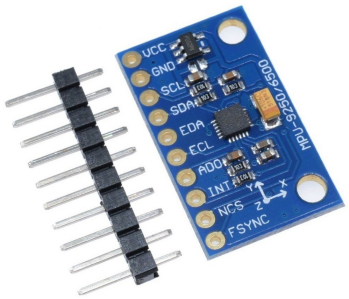


Figura 3.7: Modelo genérico del MPU9250.

le suele referir como IMU de 9 ejes. El dispositivo tiene un pequeño procesador integrado llamado DMP (Digital Motion Processor) capaz de procesar algoritmos complejos de fusión de sensores, esto es de alta importancia para el objetivo final de este proyecto. Para la segunda versión del dispositivo se utilizó el MPU-9250 fabricado por SparkFun, el cual es un poco más pequeño que el módulo genérico utilizado previamente [44].

3.2.6.3. Adafruit Feather HUZZAH ESP8266

Cómo placa de desarrollo se utilizó una Adafruit Feather Huzzah ESP8266. La misma es una placa "todo en uno" que incluye microcontrolador WiFi ESP8266 con componentes adicionales como un puerto de conectividad Micro USB y conector de baterías de 3,7V de polímero de Litio. La placa es extremadamente liviana, pesando solo 6 gramos y tiene un tamaño considerablemente pequeño (51mm x 23mm x 8mm). Respecto a conectividad al resto de los componentes, tiene integrados 9 Pins GPIO (utilizables por usuario final), que también pueden ser utilizados como I2C y SPI [45].

Como dijimos previamente, el núcleo de esta placa es un microcontrolador ESP8266 WiFi con un reloj de 80Mhz, con capacidad de ser aumentada a 160 Mhz, que funciona con un voltaje de 3,3. Compatible con los protocolos IEEE 802.11 b/g/n Wi-Fi que le permite conectividad fundamental para dispositivos IoT. Puede ser conectado por USB gracias al chip USB-Serial integrado en la placa de desarrollo que llega a velocidades de 921600 Baudios de transmisión de datos. La capacidad de

memoria de este microcontrolador es de 1MiB, pero se ve expandida por los 4MB de memoria FLASH provistos en el Adafruit Feather Huzzah. El ESP8266 tiene 16 Pines GPIO de los cuales 9 están disponibles para el uso por parte del usuario final en el Adafruit Feather Huzzah.

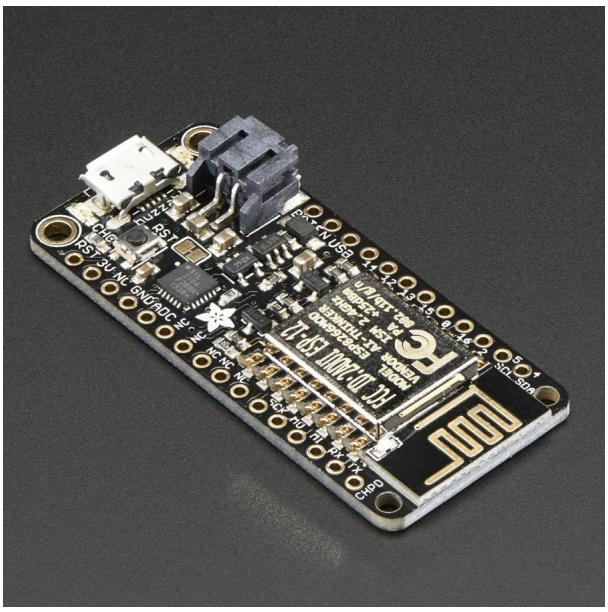


Figura 3.8: Adafruit Feather HUZZAH ESP8266.

3.2.6.4. Raspberry Pi

Una Raspberry es una computadora de placa única o "single-board computer" (SBC) del tamaño de una tarjeta de crédito y de muy bajo costo, desarrollada en el Reino Unido por la "Raspberry Pi Foundation" con la finalidad de promover la educación de ciencias de la computación en escuelas de bajos recursos y países en vías de desarrollo. La Raspberry no incluye ningún tipo de periférico, como por ejemplo teclado, mouse, ni carcasa, aunque algunos de estos pueden incluirse en paquetes tanto oficiales como no oficiales. El ecosistema Raspberry se divide en dos grandes brazos, uno es la "Raspberry Foundation", que es la encargada de promover la educación a la comunidad, y el otro es "Raspberry Pi Trading", que es quien se encarga de desarrollar los nuevos modelos y su tecnología [46].

Raspberry es un producto con propiedad registrada, lo que lleva a tener control sobre su plataforma, pero permite su uso libre tanto como para el sector educativo

como para el particular. Su software proviene del sistema operativo Debian, que fue tomado y modificado para crear su propio sistema operativo llamado Raspbian, siendo este de código libre para toda la comunidad. También permite instalar otros sistemas operativos, como por ejemplo Ubuntu MATE, Windows 10 Core, RISC OS, etc,. En todas sus versiones, la Raspberry incluye de fábrica un procesador, memoria RAM, GPU, puertos USB, HDMI, Ethernet, pines GPIO y un conector para cámara; no trae memoria de almacenamiento, es por ello que en sus primeras versiones utiliza una tarjeta SD y en sus versiones recientes una MicroSD. La Raspberry Foundation da soporte y provee distintas distribuciones para la arquitectura ARM, una arquitectura principalmente utilizada por dispositivos que funcionan a base de baterías, como teléfonos móviles, tablets, etc, debido a que estos procesadores (basados en RISC) requieren una menor cantidad de transistores que los procesadores x86 CISC, típicos en la mayoría de los ordenadores personales, llevando por lo tanto a una reducción de los costes, calor y energía. La mayoría de las Raspberry pueden ser overclockeadas para aumentar la frecuencia del reloj de la CPU. En las distribuciones basadas en Linux como Raspbian, las opciones de overclocking pueden ser aplicadas ejecutando un simple comando y sin anular la garantía. En estos casos, si la temperatura excede los 85°C se apagará automáticamente (aunque también es posible sobrescribir estas configuraciones para anular el apagado de protección por sobrecalentamiento) [47].

La Raspberry Pi Foundation a su vez, promueve Python y Scratch como lenguajes de programación principales, aunque también soporta otros lenguajes. El firmware por defecto utilizado es “closed source”, pero están disponibles distintos firmwares open source no oficiales. La comunidad entusiasta de Raspberry desarrolló una revista llamada “The MagPi” en 2015, que luego fue entregada por los voluntarios del proyecto a la Raspberry Pi Foundation.

Algunos usos de la Raspberry suelen darse para “Smart Houses” (hogares inteligentes), “Industrial Automation” (automatización industrial), productos comerciales como “Retro Gaming Machines”, “Weather Stations” (estaciones de clima), sistemas de seguridad y vigilancia, streaming de video y audio, tecnologías IoT para diversas áreas como agricultura, deporte, etc.

El desarrollo de las RPi está dividido en dos diferentes familias: la RPi original y la RPi Zero. La principal diferencia radica en que las RPi Zero son una versión

reducida de la original, en las cuales está limitada la potencia, funcionalidad y tamaño, pero a un menor costo.

- Raspberry Pi 1: es el primer modelo que salió al mercado. Está equipado con un procesador Broadcom BCM2835 de un solo núcleo con una velocidad de 700 Mhz, 256 MB de memoria RAM y una GPU Broadcom VideoCore IV. El sistema operativo debe instalarse en una memoria SD aparte y posee un solo puerto USB, sin conectividad vía Ethernet. Posteriormente fue lanzada la Raspberry Pi 1 B, que aumentaba la memoria RAM a 512 MB, 2 puertos USB 2.0 y conectividad Ethernet. Ambos modelos disponen de 8 x GPIO, SPI, I2C y UART.

Figura 3.9: Raspberry Pi 1.

- Raspberry Pi 2: es la segunda versión en salir al mercado. Posee la misma GPU que su predecesora, pero cambia el procesador por un modelo mejorado, el BCM2836 que cuenta con 4 núcleos a una velocidad de 900 Mhz. Posee la misma SDRAM de 1 GB, y cuenta con 4 puertos USB 2.0 y un puerto Ethernet 10/100 Mb. El número de pines GPIO se amplía a 17, manteniendo las funciones SPI, I2C y UART.

Figura 3.10: Raspberry Pi 2.

- Raspberry Pi 3 B: es la tercer versión de la familia y fue lanzada para mejorar la conectividad. Este modelo incluye conexión Bluetooth 4.1 y Wifi 802.11n. Fue mejorada la potencia incorporando un SoC Broadcom BCM2837 y un procesador ARMv8 de 64 bits y cuatro núcleos a una velocidad de 1.2 GHz. El GPU es el mismo que el modelo anterior (VideoCore IV), y posee 1 GB de SDRAM, un puerto Ethernet 10/100 Mb, 4 puertos USB 2.0 y 17 GPIO con funciones SPI, I2C y UART.

Figura 3.11: Raspberry Pi 3.

- Raspberry 4 B: cuarta versión de las RPi y utilizada para la tesina. Presenta un procesador Broadcom BCM2711B0 y un Cortex-A72 de 64bits y cuatro núcleos a 1.5 GHz. El GPU es reemplazado por un VideoCore VI de 500 MHz, posibilidad de optar por 1/2/4 GB de SDRAM, un puerto Ethernet, conectividad Wifi 2.4GHz/5GHz 802.11n, BLE, y Bluetooth 5.0. Posee 2 USB

2.0, 2 USB 3.0 y 2 Micro HDMI, además de contar con espacio para memoria Micro SD y USB-C para alimentación (en comparación con la alimentación Micro USB de sus predecesoras).

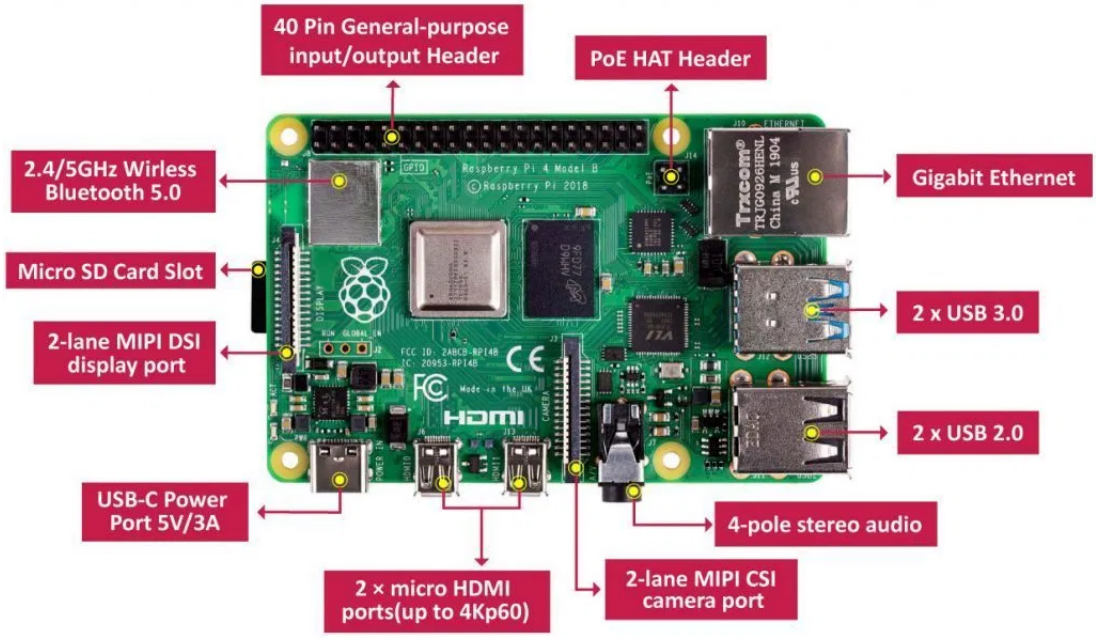


Figura 3.12: Raspberry Pi 4.

3.3. Protocolo de Comunicacion MQTT

3.3.1. Introducción

MQTT es un protocolo de conectividad $machine - to - machine$ $(M2M)$ ampliamente utilizado en *Internet of the Things* inventado en 1999 por el Dr. Andy Stanford-Clark de IBM, y por Arlen Nipper de Arcom (ahora Eurotech). Fue diseñado para transportar mensajes del estilo *publish/subscribe* y que a su vez sea extremadamente liviano. Es útil para manejar las conexiones donde se necesita transportar pequeños fragmentos de código liviano o donde la conexión de ancho de banda es limitada. Por ejemplo, es utilizado para la comunicación de sensores hacia un *broker* vía un enlace satelital, en conexiones de acceso telefónico con proveedores de atención médica, y en una amplia variedad de escenarios de automatización del hogar y dispositivos pequeños. También es ideal para aplicaciones móviles debido a

su pequeño tamaño, su bajo consumo de energía, paquetes de datos minimizados, y distribución eficiente de información a uno o varios receptores [48], [49], [50], [51].

3.3.2. Publish/Subscribe

El protocolo MQTT está basado en el principio de publicar mensajes y subscribirse a *topics* o temas ('pub/sub'), estos topics también pueden verse como canales donde los mensajes van encolándose. El comportamiento es el siguiente: varios clientes se conectan a un intermediario o broker para suscribirse a los temas que les interesan, y otros clientes también se conectan al broker y publican mensajes sobre esos topics. El broker y MQTT actúan como una interfaz en común para todo lo que desee conectarse.

3.3.3. Topics/Subscriptions

Los mensajes en MQTT son publicados en *topics*, y no hay necesidad de configurarlos, con solo publicarlos es suficiente. Los topics son tratados como una jerarquía, usando la barra (/) como separados. Esto permite un manejo sencillo a la hora de ser creados, muy similar a lo que es un sistema de archivos de un sistema operativo que se compone de directorios. Por ejemplo, múltiples computadoras pueden publicar información de la temperatura de su disco duro en un topic en común como el siguiente:

$$sensors/COMPUTER\_NAME/temperature/HARDDRIVE\_NAME$$

Los clientes pueden recibir los mensajes suscribiéndose a esos topics. La suscripción puede ser explícitamente a un topic, en el que sólo recibirá los mensajes que son publicados a ese topic, o puede incluir *wildcards* o comodines. Tenemos dos comodines disponibles: + y #.

El carácter + puede ser usado como un comodín para un solo nivel de la jerarquía. Podría utilizarse con el topic mencionado anteriormente para obtener información sobre todas las computadoras y todos los discos duros de la siguiente manera:

$$sensors/+/temperature/+$$

El carácter # puede ser usado como comodín para todos los niveles de la jerarquía. Esto significa que debe ser el último carácter en la suscripción. Por ejemplo podría ser usado de la siguiente manera, en la que recibiríamos información de la temperatura de todos los discos duros pero de una computadora en particular:

$$sensors/COMPUTER\_NAME/temperature/\#$$

Los niveles de topics de longitud cero también son validos, lo que puede conducir a un comportamiento ligeramente no obvio. Por ejemplo, un topic $a/topic$ coincide correctamente con una suscripción de $a/+/topic$. De la misma manera, existen distintos niveles de topics de longitud cero en el principio y el final de una cadena para un topic, por ejemplo $a/topic/$ coincidirá con una suscripción de $+/a/topic$, #, o /#. Y un topic $a/topic$ coincidirá con una suscripción de $a/topic/+$ o $a/topic/\#$.

3.3.4. Quality of Services

MQTT define tres niveles de *Quality of Service* (*QoS*) o calidad de servicio. La calidad de servicio define la dificultad con la que el broker intentará garantizar que se reciba el mensaje. Los mensajes pueden enviarse a cualquier nivel de QoS, y los clientes pueden intentar suscribirse a topics en cualquier nivel de QoS. Esto significa que el cliente elige la QoS máxima que recibirá. Por ejemplo, si un mensaje se publica en QoS 2 y un cliente está suscrito con QoS 0, el mensaje se entregará a ese cliente con QoS. Si un segundo cliente también está suscrito al mismo topic, pero con QoS 2, entonces recibirá el mismo mensaje pero con QoS 2. Para un segundo ejemplo, si un cliente está suscrito con QoS 2 y un cliente publica en QoS 0, el cliente lo recibirá en QoS 0.

Cuanto mayor sea el nivel de QoS más confiable será, pero implica una latencia más alta y tienen mayores requisitos de ancho de banda.

- **QoS 0**: El broker o cliente entregará el mensaje una vez, sin confirmación.

- **QoS 1**: El broker o cliente entregará el mensaje al menos una vez, con confirmación requerida.
- **QoS 2**: El broker o cliente entregará el mensaje exactamente una vez mediante un *handshake* de cuatro pasos.

3.3.5. Clean session / Durable connections

En la conexión, un cliente establece un flag de *clean session* o sesión limpia, que es normalmente conocido como el *clean start* o inicio limpio. Si el flag de clean session está establecido en false, entonces la conexión es tratada como una conexión durable. Esto significa que cuando el cliente se desconecte, las suscripciones que tenga y los mensajes subsiguientes de QoS 1 y 2 se almacenarán en el broker hasta que se conecte nuevamente en el futuro. Si el flag se clean session es true, todas las suscripciones se eliminarán para el cliente cuando se desconecte.

3.3.6. Wills

Cuando un cliente se conecta a un broker, puede informarle que tiene un *will* o testamento. Este es un mensaje que desea que el broker envíe cuando un cliente se desconecta inesperadamente. El mensaje *will* tiene un topic, un QoS y un estado igual que cualquier otro mensaje.

3.3.7. Security

MQTT puede requerir un username y password por parte de los clientes para conectarse a un broker. A su vez como MQTT corre sobre TCP, la conexión puede estar encriptada con SSL/TSL para garantizar la privacidad de la comunicación.

3.3.8. MQTT-SN

Aunque MQTT está diseñado para ser liviano, tiene dos inconvenientes para dispositivos muy limitados. Un inconveniente es que cada cliente debe ser compatible con TCP, y por lo general, mantendrá una conexión abierta al broker en todo momento. Para algunos entornos donde la pérdida de paquetes es alta o los recursos son escasos, esto es un problema. El otro inconveniente debe a que los nombres de los topics suelen ser cadenas largas que los hacen poco prácticos para el protocolo 802.15.4. Estas dos deficiencias se abordan mediante el protocolo MQTT-SN, que define una asignación entre UDP y MQTT y agrega soporta del broker para indexar nombres de topics.

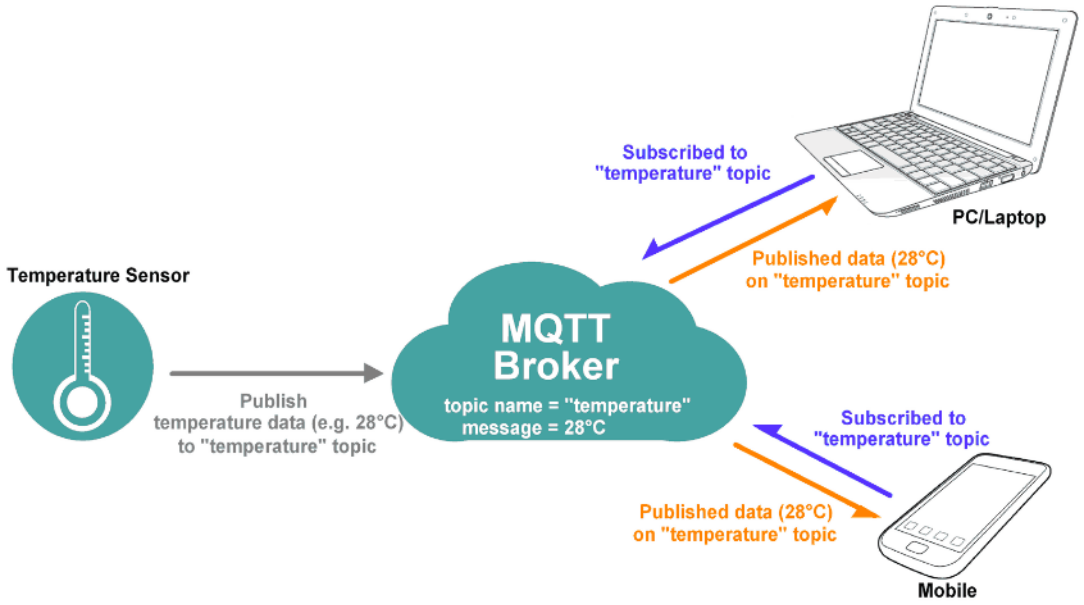


Figura 3.13: Funcionamiento del protocolo MQTT.

3.4. Software analizado

3.4.1. Arduino IDE

3.4.1.1. Introducción

El Arduino Integrated Development Environment es una aplicación multiplataforma escrito en JAVA. En él, se escribe código C o C++ que se cargará en las placas de desarrollo.

Arduino IDE provee el programa avrdude para convertir el código ejecutable en un archivo hexadecimal (en un archivo .bin) es "flasheado" (escrito) en la placa indicada, después este es cargado por el firmware de la placa. Por default se utiliza avrdude pero puede ser cambiado. Cuando escribimos código, se guarda en un archivo .ino por proyecto, que es preprocesado por el IDE y es transformado a un archivo .cpp (C++), después el archivo .ino y las librerías importadas son compiladas y linkeadas. Finalmente el archivo .bin es "flasheado" en la placa. Posee librerías específicas para múltiples tipos de placas.

Como ya mencionamos, el lenguaje Arduino es un set de funciones C/C++ que pueden ser llamadas desde el código, el "sketch" realizado es sometido a pequeños cambios y es pasado directamente al compilador C/C++, en este caso avr-g++. Esto significa que todo el estándar de C y C++ soportado por avr-g++ debería funcionar en Arduino.

No necesariamente necesitamos el Arduino IDE para programar en una placa Arduino Compatible, ya que es simplemente una placa de desarrollo AVR, se podría usar directamente AVR C o C++ (con avr-gcc y avrdude o AVR Studio) para programar en ella. Cabe destacar que las placas en las que escribimos nuestros programas no deben ser necesariamente Arduino, cualquier placa "Arduino compatible" funcionara, por ejemplo, Sparkfun, NodeMCU, Adafruit, Phoenix, etc. También cabe aclarar que no todas estas placas utilizan microcontroladores de arquitectura AVR, por lo tanto con los "cores" y librerías necesarias para descargar desde el gestor (Board Manager) que ofrece el IDE podremos manejarlas [52].

En resumen, la placa es conectada a la computadora por USB, para interactuar con Arduino IDE. Escribimos nuestro código en el IDE, el cual lo sube/flashea al microcontrolador de la placa, quien ejecuta el código, interactuando con los inputs y output tales sensores, luces, motores, etc.

3.4.1.2. Formato básico del código Arduino

Un sketch de arduino consiste en dos piezas de código principales, la función setup() y la función loop() [53].

- **setup()**: es una función que define el estado inicial de la placa de desarrollo y los dispositivos conectados a ella en el booteo (arranque) y solo corre una vez. Generalmente lo que se define es la funcionalidad de los pines, y se configuran las clases y las variables con los parámetros de inicialización.
- **loop()**: es la función que ejecuta todo sketch de Arduino cuando setup() se completa. Es la función principal y el loop que corre constantemente una y otra vez desde que prendemos el dispositivo hasta que lo apagamos, describe la lógica principal del circuito.

Otra forma de ver la lógica de un sketch de Arduino es en cuatro partes fundamentales:

- **Setup**: lo explicado previamente de la función setup, tareas que deben realizarse una única vez, como calibración de sensores e inicialización de variables.
- **Input**: en el inicio del loop lee entradas de datos de los dispositivos conectados.
- **Manipulate Data**: se transforman los datos, aplicando usualmente filtros, se les da un formato legible para enviar a otros dispositivos/programas o para ser leídos por humanos.
- **Output**: los datos salen del dispositivo por sus líneas de transmisión, puede ser por cable o puede ser por Wi-Fi.

Cabe aclarar que en la cabecera del archivo realizaremos la función de import de las librerías que deseamos usar. Las librerías se componen de archivos C++ (.cpp) que contienen código y C++(.h) las cuales definen los nombres y los parámetros de las funciones que llamaremos.

El IDE provee muchísimas herramientas para el manejo de nuestra placa, como por ejemplo un gestor de librerías, que permite descargar los set de librerías para las placas más comunes y para múltiples sensores que se conectan a ellas. Por otro lado provee manejo de opciones específicas de la placa conectada por USB/Puerto COM y un monitor de serie para poder ver e imprimir los datos que nos envía la placa, este mismo usado generalmente para debuggear.

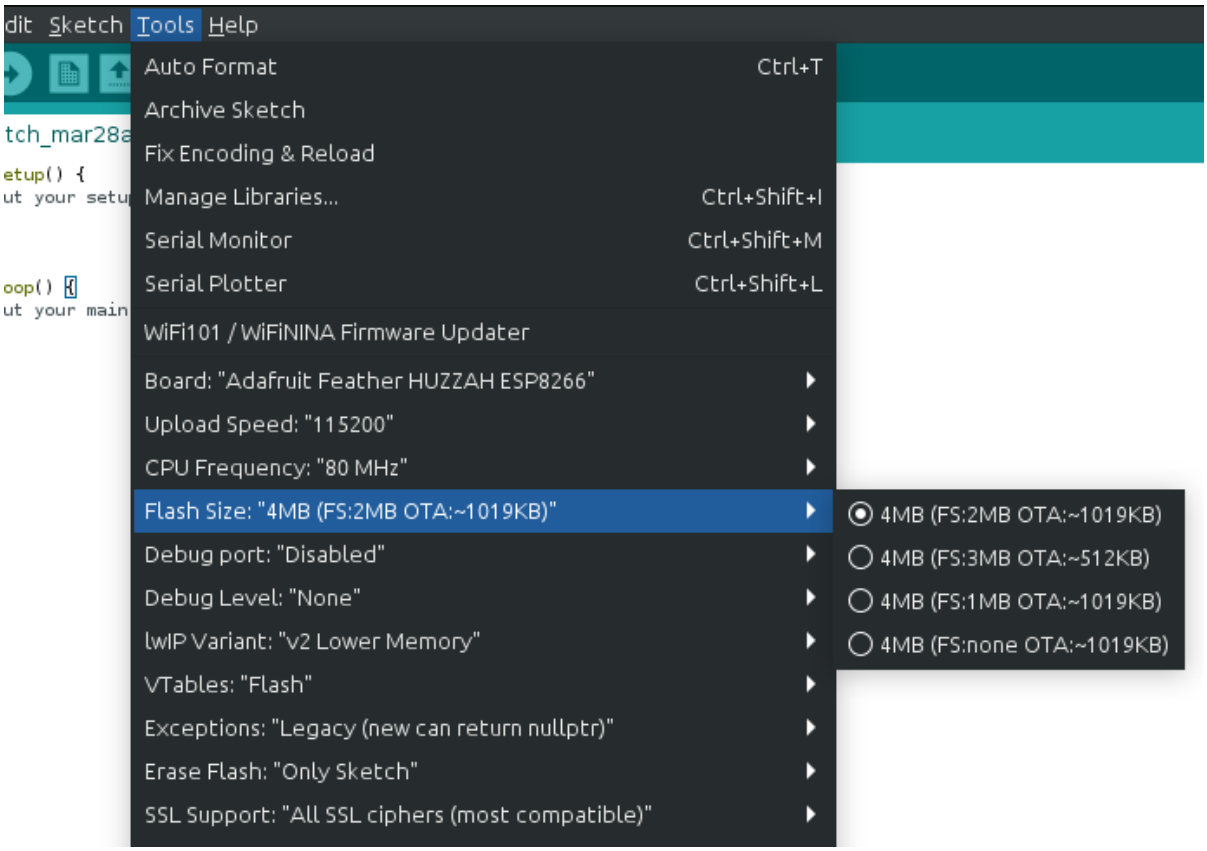


Figura 3.14: Menú con múltiples configuraciones de la placa conectada.

3.4.2. Time Series Database (TSDB)

Antes de explicar lo que es InfluxDB se explirará lo que es una base de datos de series de tiempo (TSDB) y cuál es su propósito. Una TSDB es una base de datos optimizada para "time series data" o datos con series de tiempo. Los datos de series de tiempo son simplemente mediciones o eventos que son rastreados, monitoreados, disminuyen y se agregan a lo largo del tiempo. Por ejemplo pueden ser métricas de servidores, monitoreo del rendimiento de aplicaciones, datos de la red, datos de sensores, eventos, transacciones en un mercado y muchos otros tipos de análisis de datos.

Una base de datos de series de tiempo se construye específicamente para manejar métricas y eventos o mediciones de series de tiempo. Una TSDB está optimizada para medir los cambios a lo largo del tiempo. Las propiedades que hacen que los datos de series de tiempo sean tan diferentes de otros conjuntos de datos es la gestión del ciclo de vida de los mismos, el resumen conseguido y los escaneos de grandes cantidades de registros.

Las bases de datos de series de tiempo no son un concepto nuevo, se centraban principalmente en analizar datos financieros y el comercio de acciones, pero hoy en día son muchas más las aplicaciones de este tipo de bases de datos, por ejemplo, para el IoT. Calles, automóviles, fábricas, redes eléctricas, satélites, ropa, teléfonos, microondas, todo tiene o tendrá un sensor incluido, así que básicamente todo está emitiendo un flujo incesante de métricas y eventos o datos de series de tiempo. Todo esto significa que las plataformas subyacentes deben evolucionar para admitir estas nuevas cargas de trabajo y el manejo masivo de estos, donde tenemos más objetos emitiendo datos y se necesita más monitoreo y más controles para su manejo. Es por esto que se necesitan de las bases de datos de series de tiempo que están desarrolladas específicamente para estas tareas y que sean escalables.

Una de las diferencias que distingue a las TSDB de las base de datos tradicionales es su diseño arquitectónico, esto incluye el almacenamiento y la compresión de datos con el "time-stamp" o el sello de tiempo de cada dato, la gestión del ciclo de vida de estos datos, su resumen, y la capacidad de manejar grandes escaneos y consultas dependientes de series de tiempo de cantidades de registros enormes.

Por ejemplo, con una TSDB es común solicitar un resumen de datos durante un periodo de tiempo exacto, algo que sería más complejo y costoso de lograr con un almacenamiento de claves o relacional. Las TSDB están optimizadas para estas tareas lo que proporciona tiempos de consultas a nivel de milisegundos, por ejemplo en estas bases de datos es común mantener los datos de alta precisión durante un periodo corto de tiempo, donde estos datos se agregan y disminuyen en otros datos de más largo plazo. Esto significa que por cada dato que ingresa en la base de datos, deberá tomarse una vez que finalice su periodo de tiempo. Este tipo de gestión de ciclo de vida de los datos es muy complejo de implementar para los desarrolladores de aplicaciones sobre las base de datos regulares [54], [55], [56].

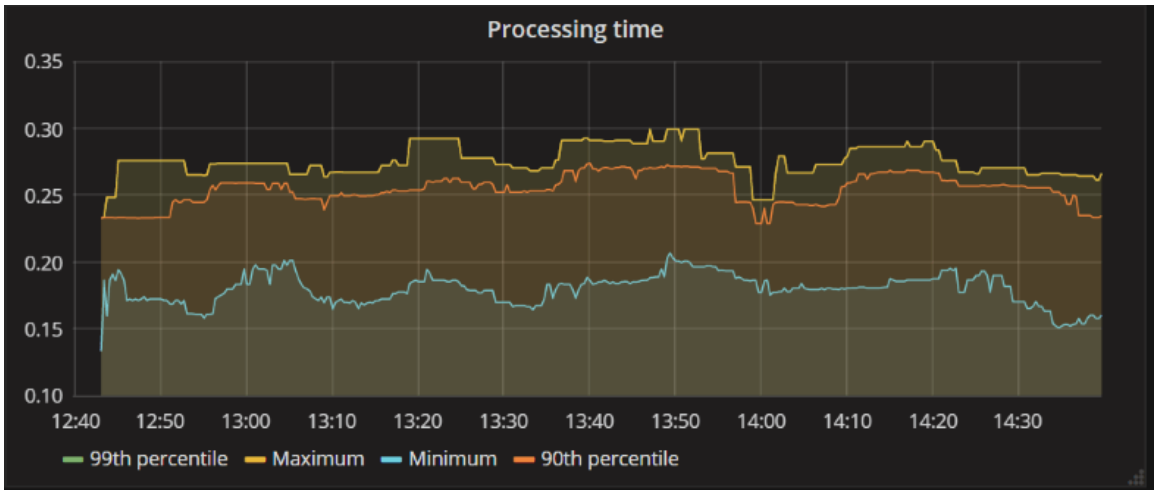


Figura 3.15: Gráfico de procesamiento de datos con series de tiempo.

3.4.2.1. InfluxDB

InfluxDB es una base de datos orientada a series de tiempo creada para procesar grandes cantidades de datos en operaciones de escritura y de lectura. Está destinada a ser utilizada como un almacenamiento de respaldo para cualquier caso de uso que involucre grandes cantidades de datos que contengan series de tiempo, por ejemplo para el monitoreo de DevOps, métricas de aplicaciones, datos de sensores IoT y análisis en tiempo real [57].

Para una mejor explicación de lo que es InfluxDB y cómo es su funcionamiento se partirá de un ejemplo claro y conciso en donde se trata de recopilar los conceptos y la terminología más importante. Se hará de cuenta que se tiene almacenada en

InfluxDB la siguiente información con la siguiente estructura:

time	butterflies	honeybees	location	scientist
2015-08-18T00:00:00Z	12	23	1	langstroth
2015-08-18T00:00:00Z	1	30	1	perpetua
2015-08-18T00:06:00Z	11	28	1	langstroth
2015-08-18T00:06:00Z	**3**	**28**	**1**	**perpetua**
2015-08-18T05:54:00Z	2	11	2	langstroth
2015-08-18T06:00:00Z	1	10	2	langstroth
2015-08-18T06:06:00Z	8	23	2	perpetua
2015-08-18T06:12:00Z	7	22	2	perpetua

Figura 3.16: Almacenamiento de información en InfluxDB.

En la figura se muestra el número de mariposas y abejas contadas por dos científicos (*langstroth* y *perpetua*) en dos ubicaciones (1 y 2) durante el periodo de tiempo comprendido entre el 18 de agosto de 2015 y el 18 de agosto de 2015. Estos datos están alojados en una base de datos llamada *my_database* y están sujetos al *autogen*, una política de retención que se explicará luego. En la figura hay una columna llamada *time* que almacena las series de tiempo y está presente por defecto en todas las bases de datos que son creadas con InfluxDB. Estas series de tiempo almacenan la fecha y la hora siguiente la RFC3339 [58] UTC asociadas con los datos de las demás columnas. Las dos siguientes columnas, $butterflies$ y *honeybees* son $field\ keys$ o claves de campo, y estos poseen un valor llamado $field\ value$ o valor de campo, que en este caso son valores enteros pero también pueden ser valores flotantes, booleanos y strings y siempre están asociados con una serie de tiempo. Los $field\ values$ en la imagen son los siguientes:

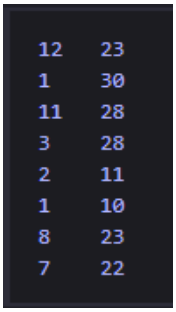


Figura 3.17: Field values almacenados en el measurement census.

La colección de pares entre *field key* y *field value* son llamados *field set*, o conjunto de campos, y en la base de datos de ejemplo se encuentran los siguientes:

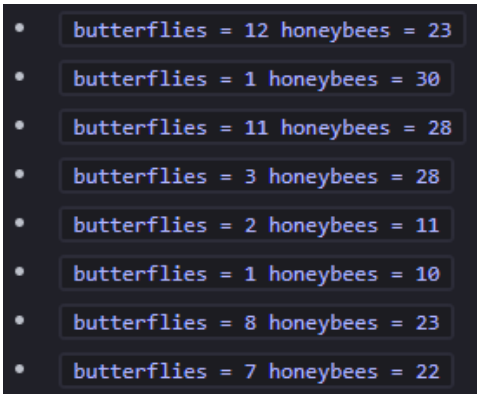


Figura 3.18: Field sets almacenados en el measurement census.

Los *field keys* son una parte necesaria dentro de la estructura de InfluxDB, no puede haber datos almacenados sin que haya ningún field key, al menos debe haber uno. También es importante tener en cuenta que estos no están indexados, por ende las consulta que usan *field values* como filtros deben escanear todos los valores que coincidan con las otras condiciones de la consulta. Como resultado, estas consultas tienen un rendimiento mucho más pobre en comparación con las consultas en los *tags* o etiquetas (que veremos en unos instantes). En general, los *field keys* no deben tener data que necesite ser muy consultada.

Las últimas dos columnas en la base de datos de ejemplo son *location* y *scientist*, y estas son *tags*. Las etiquetas están formadas por *tag keys* o claves de etiqueta y *tags values* o valores de etiqueta y ambas se almacenan como strings. Los *tag keys* en el ejemplo son *location* y *scientist*. El *tag key location* tiene dos *tag values*: 1 y 2, y

el *tag key scientist* tiene otros dos *tag values*: langstroth y perpetua. En los datos anteriores, el "tag set" o conjunto de etiquetas son las diferentes combinaciones de todos los pares clave-valor, y estos son:

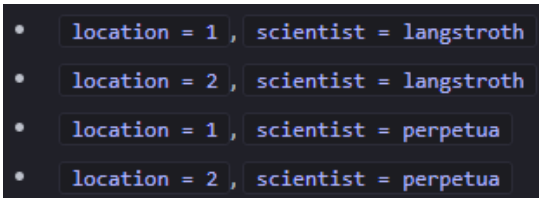


Figura 3.19: Tag sets almacenados en el measurement census.

Los *tags* son opcionales, InfluxDB no necesita tener etiquetas en su estructura de datos, pero generalmente es una buena idea utilizarlas porque a diferencia de los *fields*, los *tags* están indexados. Esto significa que las consultas en los *tags* son mucho más rápidas y que son ideales para almacenar datos comúnmente consultados. Ahora supongamos que la mayoría de las consultas se centran en los valores que poseen *honeybees* y *butterflies* que son *field keys*, como se dijo no están indexados, por lo que InfluxDB debe escanear cada valor de ambos campos afectando enormemente los tiempos de respuesta de las consultas. Para optimizar este comportamiento, puede ser beneficioso reorganizar el esquema de modo que *honeybees* y *butterflies* se conviertan en *tags*, y *location* y *scientist* se conviertan en *fields*.

time	location	scientist	butterflies	honeybees
2015-08-18T00:00:00Z	1	langstroth	12	23
2015-08-18T00:00:00Z	1	perpetua	1	30
2015-08-18T00:06:00Z	1	langstroth	11	28
2015-08-18T00:06:00Z	**1**	**perpetua**	**3**	**28**
2015-08-18T05:54:00Z	2	langstroth	2	11
2015-08-18T06:00:00Z	2	langstroth	1	10
2015-08-18T06:06:00Z	2	perpetua	8	23
2015-08-18T06:12:00Z	2	perpetua	7	22

Figura 3.20: Esquema e información reorganizada.

Ahora que butterflies y honeybees son tags, InfluxDB no tendrá que escanear cada uno de sus valores cuando realice consultas sobre sus valores, aumentando la performance y los tiempos de respuesta.

En InfluxDB existe un concepto llamado *measurement* o medición, que actúa como un contenedor de *tags*, *fields*, y la columna *time* o tiempo. El nombre que se le da al *measurement* será la descripción de los datos almacenados en los campos asociados. Los nombres de los measurements son strings, y para cualquier usuario que venga de una base de datos SQL es similar a una tabla. En este ejemplo el nombre del measurement (o tabla para un mejor entendimiento) es "census", y este nombre nos dice específicamente que los valores de los campos solo registran el número de butterflies y honeybees. Una sola medición puede pertenecer a diferentes *políticas de retención*, y esta describe cuánto tiempo InfluxDB almacena los datos ("DURATION" ó duración) y cuantos copias de estos datos se almacenan en el cluster ("REPLICATION" ó replicación). En estos datos de prueba todo lo que esté en la medición census pertenece a la política de retención *autogen*, que es creada automáticamente por InfluxDB con duración infinita y un factor de replicación establecido en uno.

Ahora que ya se explicó lo que son los *measurements*, *tags sets* y *políticas de retención*, hablaremos de lo que son las *series*. En InfluxDB las series son una *colección de puntos* que comparten un mismo measurement, tag set, y field key, y comprender el concepto de lo que es una serie es un concepto esencial a la hora de diseñar el esquema de los datos. A continuación se ve una figura donde se muestran las ocho series formadas con los datos de ejemplo:

Series number	Measurement	Tag set	Field key
series 1	census	location = 1 , scientist = langstroth	butterflies
series 2	census	location = 2 , scientist = langstroth	butterflies
series 3	census	location = 1 , scientist = perpetua	butterflies
series 4	census	location = 2 , scientist = perpetua	butterflies
series 5	census	location = 1 , scientist = langstroth	honeybees
series 6	census	location = 2 , scientist = langstroth	honeybees
series 7	census	location = 1 , scientist = perpetua	honeybees
series 8	census	location = 2 , scientist = perpetua	honeybees

Figura 3.21: Series almacenadas en el measurement census.

Otro concepto importante son los *points* o puntos, que representa un único registro de datos que tiene cuatro componentes: un measurement, un tag set, un field set, y un timestamp. Un point se identifica de manera única por su serie y su timestamp. Por ejemplo a continuación se tiene un único point:

```
name: census
-----------------
time                  butterflies honeybees   location    scientist
2015-08-18T00:00:00Z  1           30          1           perpetua
```

Figura 3.22: Point almacenado en el measurement census.

El punto en este ejemplo es parte de la serie 3 y se define por el measurement (census), el tag set (location = 1, scientist = perpetua), el field set (butterflies = 1, honeybees = 30) y el timestamp 2015-08-18T00:00:00Z.

Todos los conceptos que se explicaron se almacenan en la base de datos *my_database* que se mencionó en un principio, quien es la encargada de contener todos los usuarios, las políticas de retención, las continuas consultas, y los datos con sus series temporales [59].

3.4.3. Grafana

3.4.3.1. Introducción

Grafana es un software de visualización y análisis de código abierto, que permite consultar, visualizar, alertar y explorar sus métricas sin importan donde estén almacenadas. Brinda la posibilidad de convertir los datos almacenados en una base de datos de series temporales (TSDB) en gráficos y visualizaciones de fácil comprensión para cualquier persona [60], [61]. Grafana presenta los componentes que se explican a continuación.

3.4.3.2. Panels

Los panels (paneles) son los componentes básicos de visualización en Grafana. Cada panel tiene un Query Editor (editor de consultas) específico para el data source (fuente de datos) seleccionado en el panel. El editor de consultas de los paneles brinda la posibilidad de generar consultas del estilo SQL para obtener los datos de series de tiempo deseados.

Existen una amplia variedad de estilos y formatos para cada panel, y estos se pueden arrastrar, redimensionar y reorganizar en el Dashboard al que pertenecen. Algunos de los paneles, como el panel de gráficos, permiten graficar tantas métricas y series de tiempo como se desee, pero otros paneles requieren una reducción de una sola consulta en un solo número.

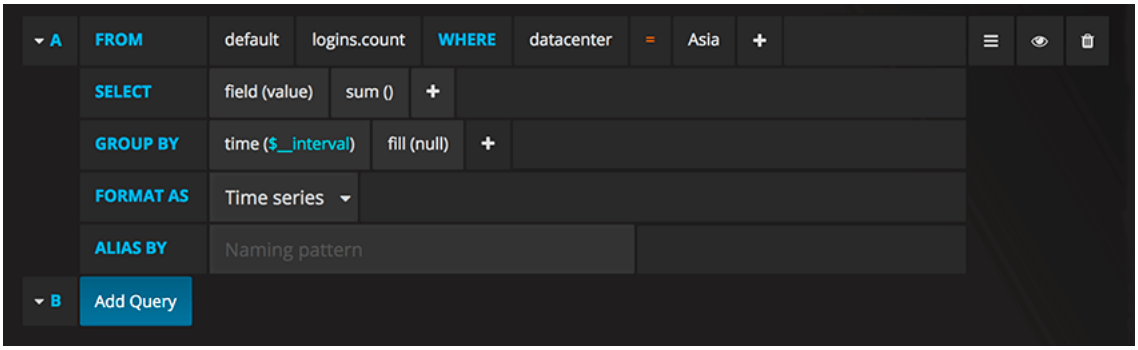


Figura 3.23: Formato del editor de consultas de un panel de Grafana similar a SQL.

3.4.3.3. Dashboards

Los dashboards (tableros) son un conjunto de uno o más paneles organizados y dispuestos en una o más filas. A su vez, los tableros permiten que cada uno de estos paneles pueden estar configurados cada uno con distintos gráficos e interactuar con los datos de cualquier data source.

Los tableros pueden refrescarse cada ciertos periodos de tiempo que servirá para actualizar los datos de cada panel con los datos traídos de la base de datos. El período de tiempo en que se refresca el tablero puede controlarse por los controles de rango de tiempo preconfigurados por Grafana, aunque también es posible crear periodos de tiempo personalizados, pudiendo hacer que el tablero completo se refresque cada 1 segundo como también cada 1 día. También pueden usar anotaciones para mostrar datos de eventos en cada panel, pudiendo ayudar a correlacionar los datos de series de tiempo en el panel con otros eventos.

```
name: census
-----------------
time                  butterflies honeybees   location    scientist
2015-08-18T00:00:00Z  1           30          1           perpetua
```

Figura 3.24: Tablero de Grafana construido por ocho paneles dedicado a medir ciertas variables del clima sobre cada panel. Los datos son capturados por sensores ubicados al intemperie.

3.4.3.4. Data Sources

Grafana puede visualizar, explorar y generar alertas con los datos de múltiples bases de datos y servicios en la nube diferentes. Cada tipo de base de datos o servicio se accede desde un Data Sources (fuente de datos) distinto, y es necesario agregar para empezar a visualizar los datos en los paneles.

Cada fuente de datos tiene un editor de consultas específico que está personalizado para las características y capacidad des que expone una fuente de datos en particular. Es posible combinar datos de múltiples fuentes de datos en un solo tablero, pero cada panel debe estar conectado a sólo una fuente de datos específica. Algunas de las fuentes de datos utilizadas por grafana son: InfluxDB, Loki, Elasticsearch, MySQL, PostgreSQL, Prometheus, AWS Cloudwatch, Azure Monitor, etc.

3.4.3.5. Logs

Junto con las métricas, Grafana permite visualizar e investigar los logs almacenados en algunas fuentes de datos como InfluxDB, Loki y Elasticsearch. Para ello, Grafana cuenta con un apartado independiente llamado "Explore" dedicado específicamente a este espacio. Los logs pueden personalizarse para seleccionar las columnas que se muestran, como también ayudar a ocultar lineas de logs duplicadas leídas de la fuente de datos.

3.4.4. Lenguajes utilizados

3.4.4.1. Python

Python es un lenguaje de programación interpretado, dinámico y multiplataforma, de alto nivel, orientado a objetos y de propósito general. Se puede utilizar para realizar aplicaciones de escritorio con interfaz de usuario, aplicaciones web, automatización de tareas e incluso en microcontroladores. Como es un lenguaje de alto nivel, permite concentrarse en la funcionalidad principal de la aplicación sin te-

ner que concentrarse en tareas complicadas como el manejo de memoria en C. Fue creado por Guido Van Rossum y lanzado por primera vez en 1991. La filosofía de diseño de Python enfatiza en la legibilidad del código por lo cual suele ser un buen lenguaje para principiantes. Se trata de un lenguaje de programación multitarea, ya que soporta distintos paradigmas de programación, como la orientación a objetos, la programación imperativa y, en menor medida, la programación funcional [62].

3.4.4.2. C++

C++ es un lenguaje de propósito general orientado a objetos (OOP) creado por Bjarne Stroustrup en 1979, siendo este una extensión del lenguaje C. Por lo tanto es posible desarrollar tanto en C++ como en C, por lo que también es considerado como un lenguaje híbrido. C++ también es considerado como un lenguaje de nivel intermedio, ya que encapsula características de alto y bajo nivel. Inicialmente, el lenguaje fue llamado "C con clases", pero fue renombrado a C++ en 1983. C++ es principalmente utilizado para programación de aplicaciones y de sistemas operativos, drivers, aplicaciones cliente-servidor y firmware embebido [63].

Capítulo 4

Propuesta del Prototipo y aporte de la tesina

A lo largo de los últimos años el estudio del mundo del deporte a través de dispositivos digitales ha crecido de manera exponencial. Esto hace que surja la necesidad de incorporar nuevas tecnologías y herramientas para obtener datos cada vez de mayor precisión a fin de obtener una mejora en el rendimiento deportivo, ya sea de manera individual o colectiva. A su vez, con la incorporación de herramientas de esta índole, es posible ayudar con la prevención de lesiones y obtener notables mejoras en el panorama táctico y estratégico de los deportistas y de equipos de distintas disciplinas.

Partiendo del enfoque anteriormente explicado, en esta tesina se decidió desarrollar una primera versión de un sistema que aporta una mejora en el estudio del deporte, brindando la posibilidad de generar datos en los entrenamientos de deportistas de alto rendimiento como también de deportistas aficionados interesados en obtener exhaustivas métricas de fácil lectura con respecto a sus entrenamientos y estado físico actual. A su vez, este trabajo contribuye un aporte de valor a toda la comunidad del deporte y a estudiantes avanzados que aspiran a involucrarse en el mundo emergente de *Internet de las Cosas*. Al ser un trabajo que involucra una gran cantidad de conceptos vistos en la carrera y en carreras de la misma índole, se espera que sirva como referencia para cualquier otra persona que esté interesada en entender ya sea aspectos de hardware, electrónica, matemática, protocolos de comunicación,

redes, base de datos, software y programación, o que simplemente desee ver cómo estas piezas se comportan en conjunto con el fin de visualizar resultados reales, fiables y de gran similitud con los obtenidos por dispositivos privativos y de mayor costo utilizados por atletas y equipos de elite.

Al finalizar esta tesina se espera contar con una primera versión de un sistema de monitoreo en tiempo real del rendimiento de deportistas en cuanto a sus entrenamientos y/o competencias, con la posibilidad de ser utilizado en diversas disciplinas del deporte como fútbol, rugby, hockey, etc. Se espera que la información calculada por los algoritmos matemáticos tras la captura de los datos por parte de los sensores, sea lo suficientemente fiable para mostrar mediante gráficos métricas precisas que sirvan de gran ayuda para los deportistas y entrenadores, permitiendo obtener una gran ventaja en cuanto a la planificación estratégica de entrenamientos y competencias, que con el transcurso del tiempo el nivel de exigencia de estas es cada vez más demandante.

A continuación se describe una lista donde se detalla, en lineas generales, algunos de los aportes realizados por la tesina:

1. Un sistema para equipos de deporte de alta competencia con bajo presupuesto que no desean gastar grandes cantidades en dispositivos de alto costo. Necesidad de incrementar su performance analizando los entrenamientos y capacidades de cada jugador midiendo métricas como cantidad de sprints, duración total en sprint, velocidad máxima, distancia recorrida, aceleraciones/desaceleraciones, posicionamiento GPS trazando el recorrido en todo momento, velocidad promedio, etc.

2. Sistema para deportistas aficionados con la curiosidad de comenzar a obtener métricas de sus entrenamientos e incursionarse en el estudio del mundo del deporte para mejorar su rendimiento y lograr mayores progresos.

3. Sistema base y de referencia con posibilidad de ser estudiado y extendido por las personas o comunidad interesada en ampliar la funcionalidad del trabajo y/o adaptarlo a sus objetivos y necesidades.

4. Objeto de estudio por estudiantes avanzados de informática o de diferentes disciplinas afines como ingeniería electrónica o matemática donde pueden

apreciar cómo los componentes de este trabajo se comunican entre sí en un aplicativo real, garantizando resultados útiles que servirán como objeto de estudio para estudiantes de otras áreas como educación física e incluso medicina aplicada al deporte.

Por último, se presenta a modo de imagen un modelo donde se muestran las partes y componentes que conformarán al sistema junto con la comunicación para lograr su funcionamiento.

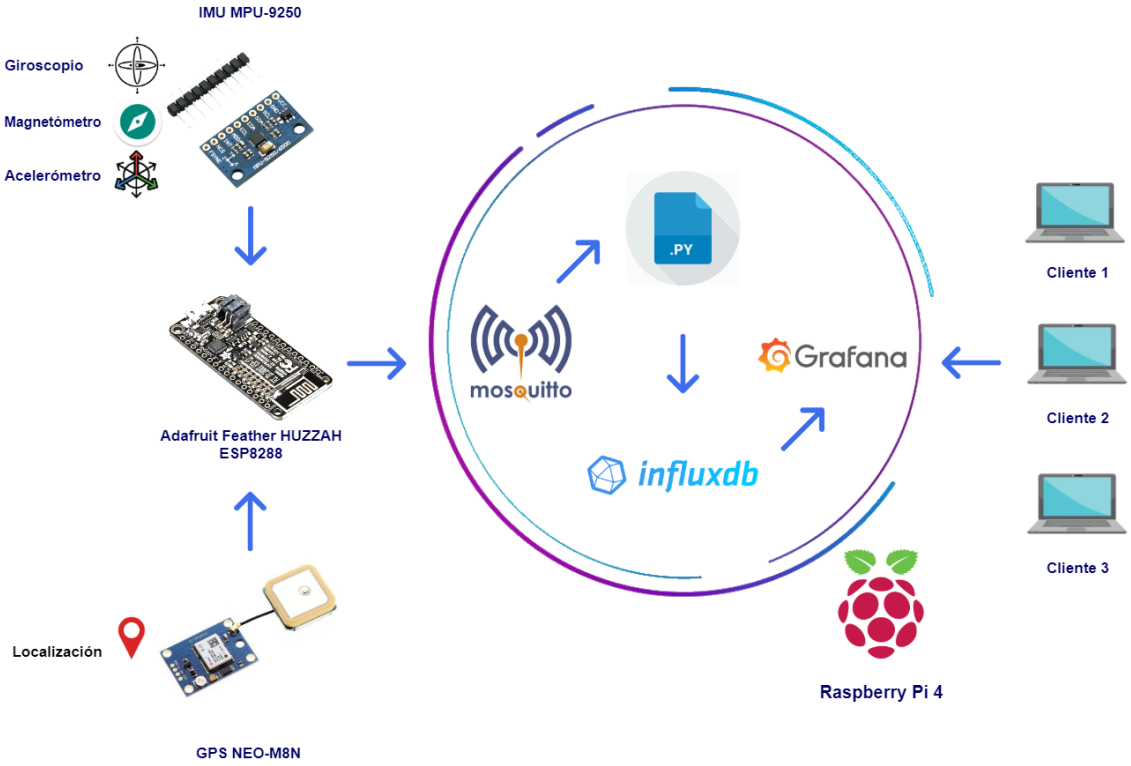


Figura 4.1: Workflow planteado del funcionamiento completo del sistema.

Capítulo 5

Desarrollo del Prototipo

En esta sección se explica cómo se inició el desarrollo del dispositivo, siguiendo el Roadmap que se utilizó para crear el mismo, lo cual ayudará a la compresión desde la base del proyecto.

5.1. Conexión del Hardware

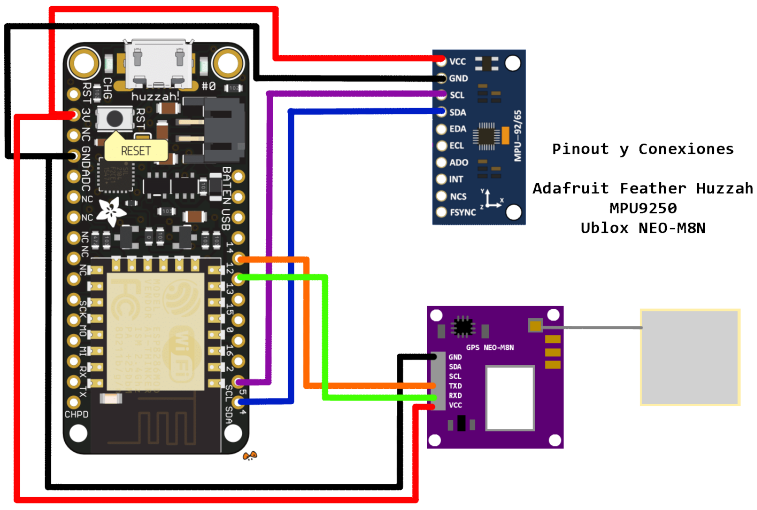


Figura 5.1: Esquema de conexión de los sensores con el Adafruit.

En la figura 5.1 podemos ver un esquema eléctrico simplificado de las conexiones que se utilizan en el dispositivo final. Inicialmente la conexión fue probada en una placa de pruebas (Protoboard o Breadboard en ingles). Las conexiones se realizan de la siguiente forma:

1. **Adafruit y MPU9250**.
    - **GND a GND**: el pin GND (Ground) es toma a tierra del dispositivo, el cual sirve para cerrar el circuito y como punto de referencia 0V (voltios) para medir el voltaje.
    - **VCC a 3V**: pin de 3V (voltios) con el cual el Adafruit alimenta al MPU9250 por el pin VCC el cual es la entrada de alimentación.
    - **SCL a SCL**: corresponde a la señal de clock para la comunicación I2C.
    - **SDA a SDA**: corresponde a la linea de comunicación maestro esclavo de la comunicación I2C.
2. **Adafruit y Ublox Neo-M8N**.
    - **GND a GND**: pin GND, sirve para el mismo propósito que explicamos previamente.
    - **VCC a 3V**: pin de 3V (voltios) para alimentar el modulo GPS.
    - **TXD a Pin 12**: TXD es la linea de transmisión del modulo GPS, por esta linea se enviarán los datos recibidos del satélite al Pin 12 del Adafruit, que en esta placa es el equivalente al pin MISO (Master In Slave Out de SPI).
    - **RXD a Pin 13**: RXD es la linea de recepción de datos del modulo GPS, por esta linea podremos cambiar la configuración del mismo enviando los datos por pin 13 del Adafruit, que en esta placa equivalente al pin MOSI (Master Out Slave In).

Una vez confirmado el funcionamiento correcto de los sensores y el dispositivo, se pasa a la siguiente etapa del prototipo en la cual, el esquema se replica en una placa de circuito perforada (Perfboard). En esta placa los dispositivos están encastrados y sus cables soldados, lo cual para la naturaleza de este proyecto es de extrema necesidad para asegurarnos de que el acelerómetro se vera afectado por la menor cantidad de vibraciones posibles.

5.2. Introducción a las librerías y pruebas de Conexión

Iniciamos probando por separado la conexión de cada uno de los dispositivos. Por cada uno de los dispositivos, se hizo una prueba simple de comunicación y uso de su librería antes de ser integrados con el resto, para entender las particularidades de cada una.

5.2.1. Adafruit y GPS uBlox NeoM8

Para poder conectarse al GPS y leer los datos del mismo es necesario entender el protocolo NMEA y la forma en la que el dispositivo envía los datos. Algunos de ellos tienen incluso sus librerías propietarias u opensource para su protocolo, como uBlox. Se decidió utilizar el protocolo NMEA ya que en nuestro caso no se necesitaban ninguna de las prestaciones particulares que uBlox ofrece sobre NMEA. Debido a que el set de datos que debíamos recuperar del GPS eran los mínimos necesarios, es decir Longitud, Latitud y Velocidad, se investigaron múltiples librerías teniendo en mente que deben ser ligeras, con una interfaz de conexión simple (evitando entrar en detalles de electrónica) una gran abstracción para la lectura de datos y recientemente actualizada. Además de todo esto se intentó que fuera una librería ampliamente conocida en el ambiente de Arduino, por lo tanto se evaluó entre las listadas oficialmente por el Arduino IDE.

La librería utilizada fue TinyGPS++ [64]. Según su desarrollador, Mikal Hart, es una librería compacta, de fácil uso para extraer datos como posición, fecha, hora, velocidad, altitud y curso de GPS para consumidores finales.

TinyGPS++ crea un objeto de tipo TinyGPS++, la librería parsea las sentencias NMEA y popula los campos del objeto con los datos parseados. Los campos del objeto y su descripción son:

- **Location**: el último fix con posición.

- **Date**: el último fix de fecha, formato UT.
- **Time**: el último fix de hora, formato UT.
- **Speed**: velocidad actual en tierra.
- **Course**: curso actual.
- **Altitude**: último fix de altitud.
- **Satellites**: el número de satélites al cual el GPS está conectado.
- **HDOP**: dilución de precisión.

Cada vez que recuperamos un dato del objeto, se recomienda utilizar el método $isValid()$ para asegurarnos que la comprobación de checksum del fix sea correcta y además el dato sea el último leído del dispositivo, debido a que no todos los campos del objeto son actualizados al mismo tiempo, nos puede quedar parte del objeto con datos del fix anterior y otros con el fix nuevo. Para evitar problemas con esto, es utilizado el método $isValid()$.

El código de los programas en Arduino consiste generalmente en la importación de librerías, la inicialización de variables en la función *setup*() y el loop principal. Recordando esto podemos ver que el siguiente código respeta el formato.

```
1  #include <TinyGPS++.h>
2  #include <SoftwareSerial.h>
3
4  static const int RXPin = 12, TXPin = 13;
5  static const uint32_t GPSBaud = 9600;
6
7  // TinyGPS++ object
8  TinyGPSPlus gps;
9
10 // Conexion Serial al dispositivo
11 SoftwareSerial ss(RXPin, TXPin);
12
13 void setup()
14 {
15   Serial.begin(115200);
16   ss.begin(GPSBaud);
```

```
17   Serial.println(F("Inicializando Ejemplo"));
18 }
19
20 void loop()
21 {
22   while (ss.available() > 0)
23     if (gps.encode(ss.read()))
24     displayInfo();
25     if (millis() > 5000 && gps.charsProcessed() < 10)
26     {
27         Serial.println(F("GPS No detected. Check Wiring."));
28         while(true);
29     }
30 }
31
32 void displayInfo()
33 {
34   Serial.print(F("Location: "));
35   if (gps.location.isValid())
36   {
37     Serial.print(gps.location.lat(), 6);
38     Serial.print(F(","));
39     Serial.print(gps.location.lng(), 6);
40   }
41   Serial.println();
42 }
```

Para conectarnos al dispositivo debemos iniciar una conexión de SoftwareSerial, indicando los pines de Recepción (Rx) y Transmisión (Tx), estos dependen de cómo conectamos los cables al armar el dispositivo. En nuestro fueron el pin 12 como Rx y 13 como Tx. En el setup iniciamos las dos conexiones serial, primero de nuestro PC al dispositivo, con un Baudrate 115200 y después del dispositivo al GPS, que por su parte utiliza un baudrate mucho menor, este es configurable, aunque por defecto es 9600 y es el recomendado.

La función Loop contiene un while que lee datos del dispositivo siempre que el Adafruit detecte que el GPS está listo para enviar datos, los lee del serial y imprime con la función $displayInfo()$ que verifica, formatea el string de forma entendible para lectura humana y los imprime. Si durante un tiempo determinado no hemos

recibido más de 10 chars del GPS podemos asumir que hubo un fallo en la conexión. El resultado final de grabar el código en el dispositivo, iniciarlo y recibir los datos por el Serial Monitor correspondiente se puede ver en la figura 5.2. En la misma vemos su gran velocidad, tomando más de 12 muestras en menos de un segundo, y como en ese segundo no nos movimos el cambio es mínimo (por posibles errores de precisión).

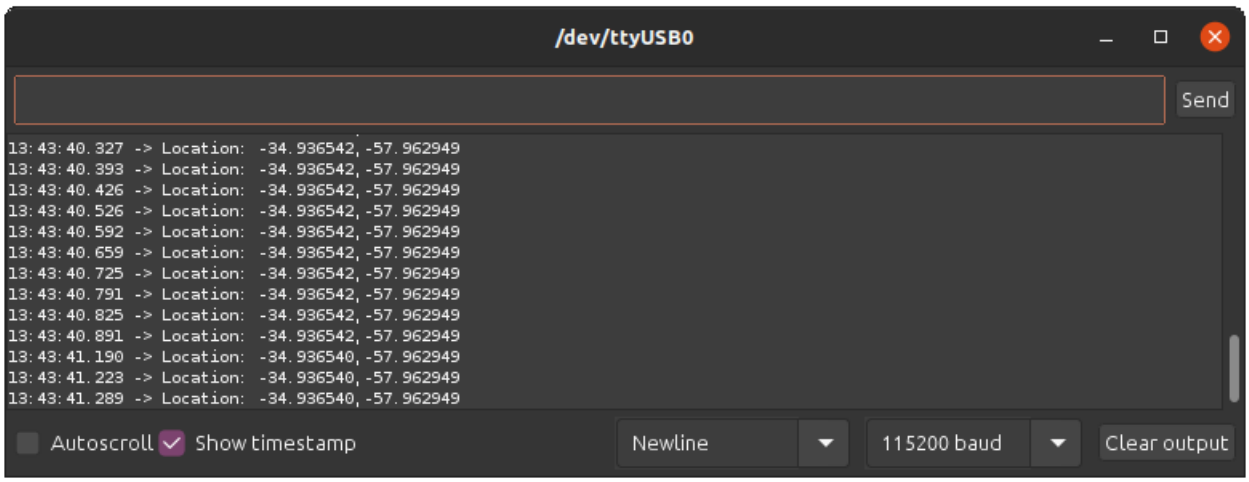


Figura 5.2: Envío de latitud y longitud por Serial.

5.2.2. Adafruit y MPU9250.

Para la elección de la librería utilizada en el caso del MPU9250 y el Adafruit se tuvieron en cuenta otros factores. Debido a la gran customización que necesitábamos buscamos una librería que nos permitiera cambiar los valores de los registros acelerómetro, manipular el tipo de conexión, y evitar problemas de performance. La librería utilizada fue la oficial de Sparkfun MP9250, esto también se debe a que parte de los MPU9250 utilizados en este proyecto eran fabricados por Sparkfun, además de funcionar muy bien con los dispositivos genéricos. La librería está en constante desarrollo y con soporte por parte de los desarrolladores [65].

```
1 #include <SoftwareSerial.h>
2 #include "MPU9250.h"
3
4 #define MPU9250_ADDRESS  0x68
5
```

```
MPU9250 IMU;

void setup() {
  Serial.begin(38400);
  initIMU();
}
```

Pasaremos por alto el contenido en detalle de la función $initIMU()$. La librería utilizada requiere seguir un protocolo de inicialización complejo de bajo nivel. Lo que debemos saber de ella es que realiza el proceso de testeo de hardware, registros del MPU, inicialización y configuración del mismo. Diferente del GPS, el MPU9250 se conecta con el Adafruit mediante el protocolo I2C indicando la dirección de memoria en hexadecimal en la cual se ubica, en este caso 0x68. En la función initIMU se inicializa el proceso de validación de conexión leyendo el registro WHO_AM_I_MPU9250 y esperando el valor 0x71. Si este es correcto, quiere decir que nuestra conexión de hardware es exitosa y prosigue con el testeo y configuración.

Como mencionamos previamente, esta librería tiene un nivel de abstracción más bajo que la utilizada para el GPS. Por lo cual debemos recuperar los datos y hacer operaciones para representarlos de la manera que necesitamos.

```
void loop() {
  if (IMU.readByte(MPU9250_ADDRESS, INT_STATUS) & 0x01)
  {
    IMU.readAccelData(IMU.accelCount);
    IMU.ax = (float)IMU.accelCount[0] * IMU.aRes;
    IMU.ay = (float)IMU.accelCount[1] * IMU.aRes;
    IMU.az = (float)IMU.accelCount[2] * IMU.aRes;

    IMU.readGyroData(IMU.gyroCount);
    IMU.gx = (float)IMU.gyroCount[0] * IMU.gRes;
    IMU.gy = (float)IMU.gyroCount[1] * IMU.gRes;
    IMU.gz = (float)IMU.gyroCount[2] * IMU.gRes;

    IMU.readMagData(IMU.magCount);
    IMU.mx = (float)IMU.magCount[0] * IMU.mRes * IMU.
    factoryMagCalibration[0] IMU.magBias[0];
    IMU.my = (float)IMU.magCount[1] * IMU.mRes
                * IMU.factoryMagCalibration[1] - IMU.magBias[1];
```

```
18      IMU.mz = (float)IMU.magCount[2] * IMU.mRes
19                  * IMU.factoryMagCalibration[2] - IMU.magBias[2];
20
21    }
22    String data = "";
23    data += "AX= "+String(IMU.ax,6)+",";
24    data += "AY= "+String(IMU.ay,6)+",";
25    data += "AZ= "+String(IMU.az,6)+"";
26    Serial.println(data);
27
28  }
```

Después recuperar los datos del dispositivo, multiplicarlos por la resolución del sensor y eliminar el valor de desviación promedio que se obtiene al calibrar, podemos imprimir los datos al Serial Monitor para visualizarlos. Debemos tener en claro que por más que el dispositivo esté una superficie plana como una mesa, factores como el desnivel de la tierra, las pequeñas vibraciones en el ambiente, el posible error del sensor y de calibración pueden generar que los valores no sean 0 estando en reposo, por lo cual los resultados en la siguiente imagen son completamente normales. Por otro lado, sobre el eje Z vemos el efecto de la fuerza de gravedad, con un valor que se aproxima a 1G, lo que es igual a 9,8m/s. Moviéndose sobre cada uno de los ejes de tal manera que la fuerza de gravedad caída sobre ellos podemos probar que este funcionando correctamente, en ese caso por cada eje que probemos el valor del mismo va a ser 1G.

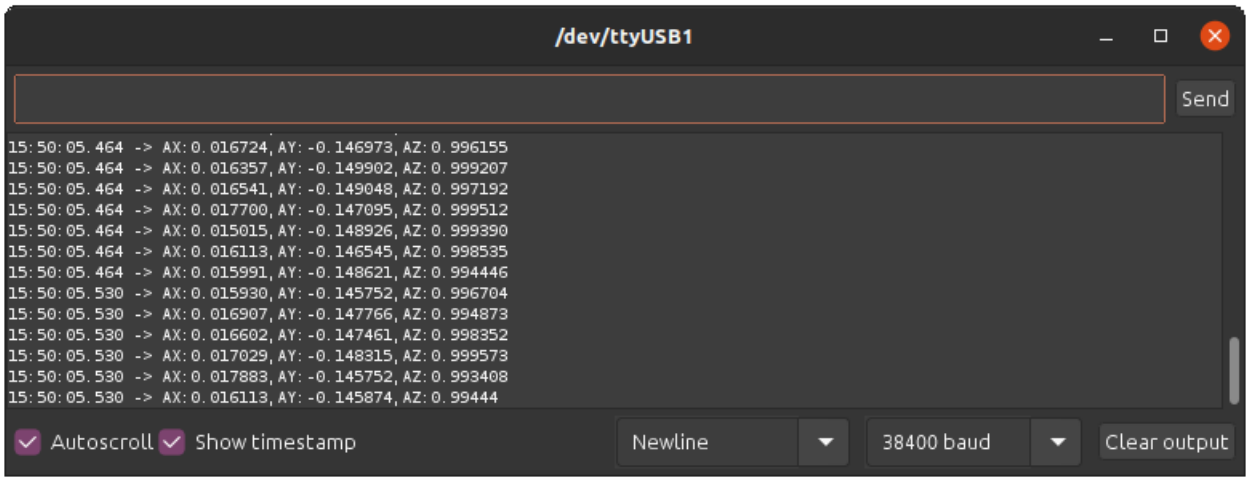


Figura 5.3: Resultados de los tres ejes del acelerómetro por Serial.

5.2.3. Conexión WiFi del Adafruit y PC

Debido a que gran cantidad de dispositivos IoT utilizan conexión WiFi, el manager de librerías de Arduino provee soporte para cientos de módulos. Arduino provee una librería oficial para su dispositivo Arduino UNO WiFi que utiliza el módulo ESP8266 para su conectividad wifi. Debido a que la misma ofrece más prestaciones de las que necesitamos y debemos mantener el proyecto lo más limpio y liviano posible, se decidió usar la librería oficial de la comunidad del ESP8266, llamada ESP8266WiFi.

Para establecer una conexión utilizando esta librería, debemos proveer al método $WiFi.begin()$ el nombre de la red y la contraseña. Lamentablemente en esta versión del dispositivo no es posible modificar los datos de red, el código debe ser escrito otra vez en el mismo para realizar estos cambios. La librería provee la función $WiFi.status()$ la cual utiliza los siguientes valores para describir el estado de la conexión WiFi:

- 0 : WL_IDLE_STATUS se obtiene cuando se está cambiando entre estados.
- 1 : WL_NO_SSID_AVAIL cuando el SSID configurado es inalcanzable físicamente.
- 3 : WL_CONNECTED se obtiene cuando se está cambiando entre estados.
- 4 : WL_CONNECT_FAILED si la password es incorrecta.
- 6 : WL_DISCONNECTED si no está configurado en modo station.

El ESP8266 puede funcionar en múltiples modos, el modo que estaremos usando, el cual es el default, es el modo STA (Station) en el cual el dispositivo sólo será un dispositivo más conectado a un Access Point indicado, como cualquier PC o Smartphone que se conecte a una red. El otro modo es el AP en el cual el ESP se convierte en un Access Point para que los dispositivos se conecten a el, de este modo no tendríamos conectividad a internet. Cuando el dispositivo está en modo STA es capaz de reconectarse a la red automáticamente si hubo algún problema de conexión de por medio. Si nos vamos del rango del Access Point, este pasara al

estado WL_NO_SSID_AVAIL pero volverá a conectarse al volver a estar dentro del rango.

```
#include <ESP8266WiFi.h>

void setup()
{
  Serial.begin(115200);
  Serial.println();

  WiFi.begin("network-name", "pass-to-network");

  Serial.print("Connecting");
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println();

  Serial.print("Connected, IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  WiFiClient client;
  if (!client.connect("192.168.1.40", 1337)) {
    Serial.println("Error de conectividad");
    delay(5000);
    return;
  }
  Serial.println("Enviando data al servidor");
  if (client.connected()) {
    client.println("Hola! Soy el ESP8266");
  }
}
```

En el loop principal del código, creamos una instancia de la clase cliente, de este modo el ESP8266 se convertirá en un cliente del servidor el cual le indicaremos como parámetro al llamar a client.connect(ip, puerto) con la dirección IP y el Puerto al que nos queremos conectar. En la última foto vemos el resultado de la ejecución del

código de prueba, se puede ver que recibimos exitosamente el mensaje que envía el ESP y los mensajes de log por el Serial Monitor. El error de conexión que recibimos al final se debe a que el netcat cerró la conexión inmediatamente al haber recibido el mensaje.

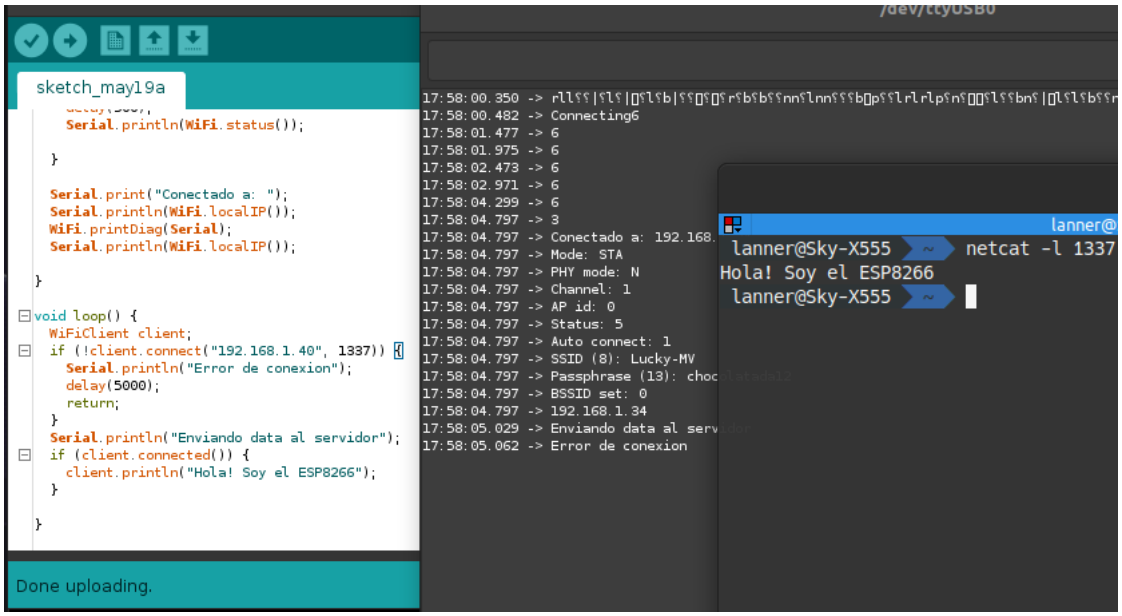


Figura 5.4: Envío de datos desde el dispositivo a un puerto en escucha en la PC mediante el ESP8266.

Para la conexión de cada dispositivo a la Raspberry utilizaremos el protocolo MQTT, que corre sobre TCP, por lo tanto la forma de establecer conexión a un puerto e IP indicados va a variar y se agregara complejidad propia del protocolo.

5.3. Integrando las tres conexiones

Al realizar las pruebas de las tres conexiones y verificar que las mismas transmiten los datos de manera correcta se empezó con el desarrollo del programa unificado para grabar en el dispositivo y hacer la prueba de integración. Para el mismo se planificó la siguiente estructura lógica:

1. Declaración de variables necesarias para conexión y cálculos matemáticos.

2. Funcion setup() con la inicialización de los dispositivos y conexiones:

    a) Se inicializa el MPU9250 junto con el AK8963. El mismo necesita unos segundos de calibración.

    b) Se realiza la conexión WiFi, se hace una sola vez debido a que el módulo ESP8266 tiene la capacidad de reconectarse por su cuenta.

3. Funcion loop() con la lógica principal del programa:

    a) Conexión al server MQTT, aunque la conexión WiFi se realice en el setup, la conexión MQTT se debe verificar y reintentar en el loop en caso que falle, ya que a diferencia de la conexión WiFi, esta no vuelve a establecerse si ocurre una falla.

    b) Lectura de los datos del GPS si es que están disponibles. El GPS recibe los datos mucho más lento que el IMU. Se necesita preguntar constantemente si está listo para enviar datos y si los mismos son nuevos. No nos aporta precisión repetir fixes y aumentaría la carga del dispositivo.

    c) Lectura de los datos del acelerómetro.

Los pasos indicados en el inciso 3 se repiten constantemente hasta que el dispositivo es apagado. Dejando el setup() y loop() de la siguiente forma:

```
1
2  void setup()
3  {
4    Serial.begin(38400);
5    wifiSetup();
6    imuSetup();
7    count_delta = millis();
8  }
9
10
11 void loop()
12 {
13   connectMQTT();
14   getGPSReadings(10);
15   calculateAccelReadings();
16 }
```

Para esta etapa de pruebas solo se enviaron los datos en crudo del dispositivo a recibir en el broker MQTT [66].

5.4. Broker MQTT.

5.4.1. Conectando con el broker

Para conectar el dispositivo al broker se utiliza la siguiente función:

```
1
2  void connectMQTT()
3  {
4
5    while (!client.connected())
6    {
7      Serial.println("Connecting to MQTT...");
8
9      if (client.connect(device_id))
10     {
11         String msg;
12         msg += device_id + "," + String("Connected to MQTT Server");
13         sendTopicData("/logs", msg);
14         fallbackFunction();
15     } else
16     {
17       Serial.print("Error, status:  ");
18       Serial.print(client.state());
19     }
20   }
21 }
```

Esta función es llamada dentro de la función loop() mencionada anteriormente, donde *client* queda definida por las instrucciones:

```
1 WiFiClient client;
2 PubSubClient mqttClient(client);
```

La responsabilidad de esta función es conectarse con el servidor MQTT gracias a la declaración de las siguientes constantes:

```
const char* ssid = "deportes_wifi";
const char* password =  "strongpassword";
const char* mqttServer = "192.168.x.x";
const int mqttPort = 1883;
```

Si logra conectarse, enviará un mensaje por el topic */logs* que se verá reflejado en el broker, y en caso de no ser posible, hará un delay(1000) y volverá a intentar establecer la conexión. Una vez establecida la conexión, los datos podrán enviarse hacia el broker.

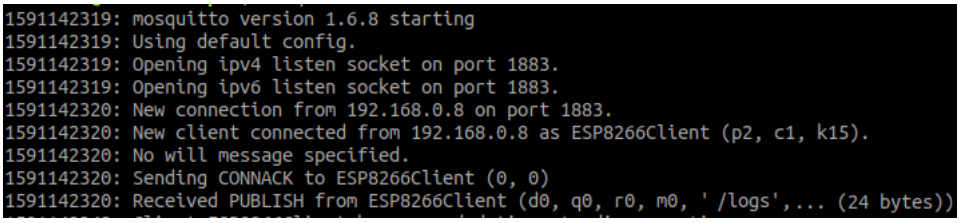


Figura 5.5: Conexión MQTT al broker.

En la figura 5.5 puede verse como se establece una nueva conexión en el broker (con el puerto 1883, puerto por defecto para el protocolo MQTT) por parte del dispositivo ESP8266, seguido de un mensaje hacia el topic */logs* que veremos a continuación. También, más adelante hablaremos de los mensajes intercambiados entre el dispositivo y el broker para establecer una conexión estable.

5.4.2. Enviando los datos al broker

Para enviar los datos desde el dispositivo al broker se ha decidido utilizar la librería *"PubSubClient"*, una librería simple que permite generar mensajes del estilo publish/subscribe con un servidor que soporte MQTT. Todos los mensajes enviados se publicarán con QoS 0, y este cliente utiliza la versión 3.1.1 de MQTT por defecto.

```
void sendTopicData(const char* topic, String msg)
{
    strcpy(buffer,msg.c_str());
    mqttClient.publish(topic,buffer,false);
    memset(buffer, 0, sizeof buffer);
}
```

Mediante esta función, se envían los datos provenientes de los sensores de los dispositivos hacia el broker MQTT Mosquitto. Esta función es una "helper function" (invocada por las otras funciones) que recibe el topic al cual deben enviarse los datos, y el mensaje en formato String. Las tres lineas corresponden respectivamente a:

- Copiar el mensaje recibido por la función en la variable buffer.
- Publicar el mensaje en el topic especificado.
- Limpiar la variable buffer.

Cada mensaje que se envía al broker tiene sus campos separados por coma, los dispositivos actualmente no tienen la capacidad de obtener un device_id en el arranque, por lo cual deben tener previamente un ID en el código (en este caso la variable device_id), con el mismo el sistema corriendo en el servidor MQTT generara los measurement correspondientes para cada dispositivo y sus datos enviados, ya sean logs, GPS o IMU. Luego se mostrara de manera más completa como el dispositivo envía los datos calculados, como el broker los recibe seguido de cómo se procesan esos datos con Python, y cómo se almacenan en InfluxDB para luego ser mostrados mediante distintas métricas calculadas en Grafana.

5.5. Calculando la Aceleración y la Velocidad

5.5.1. La problemática de los datos en crudo

Como se explicó previamente en el marco teórico, para realizar los cálculos de velocidad se necesita la aceleración lineal. El acelerómetro mide la aceleración de tres ejes en m/s2. Este devuelve la aceleración aplicada al dispositivo midiendo la fuerza que se aplica sobre el sensor. Esta aceleración medida siempre está afectada por la fuerza de gravedad de la tierra:

$$a = -g - \frac{\sum F}{m}$$

Donde a es la aceleración aplicada al dispositivo, g es la fuerza de gravedad, F es la fuerza que actúa sobre el dispositivo, y m es la masa del dispositivo. El signo representa la sumatoria de los ejes x, y, z.

Así, la única forma de que el dispositivo nos devuelva $0\,\mathrm{m/s^2}$ en sus tres ejes es cuando esta en caída libre acelerando hacia el piso. En esta situación, los objetos parecen no tener peso, debido a esto la masa dentro del acelerómetro no causa ningún tipo de desplazamiento en sus mecanismos internos, lo cual genera el valor cero. Las leyes de Newton muestran que un cuerpo en caída libre es un sistema inercial tal que la suma de las fuerzas gravitacionales e inerciales es igual a cero. Por lo tanto, el dispositivo colocado sobre una superficie plana debería indicar:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix}$$

Siendo g la gravedad de la tierra $9.806\,65\,\mathrm{m/s^2}$. Por ende debemos buscar la forma de eliminar la gravedad. Con esto nos quedan claros los siguientes puntos:

- Al dejar el dispositivo en reposo, la aceleración a es $a = g$.

- En caída libre, la aceleración a es $a = 0$
- Si cambiamos la orientación del dispositivo en el espacio y queremos medir su aceleración en sus ejes, debemos remover la aceleración por gravedad de las lecturas del sensor, a esto se lo conoce como *Gravity Compensation*.
- Al rotar el dispositivo, el efecto de la gravedad sobre su masa interna cambiará.

En el caso del giroscopio, los datos obtenidos del mismo están expresados en grados/segundos (grados sobre segundos) y como se mencionó previamente, indican la razón de rotación sobre los ejes x, y z, los cuales serán positivos si se rota en el sentido de las agujas del reloj y negativos en sentido opuesto. Los tres ejes del dispositivo deben marcar una magnitud de 0 grados por segundo cuando el mismo está en reposo sin ningún tipo de movimiento. Para obtener el ángulo del giroscopio se debe realizar una integración simple, la cual también convierte el ruido en drift (desviación), esta desviación puede ser compensada con una mayor frecuencia de sensor y con la correcta calibración del mismo, aunque no puede ser eliminada completamente [67].

5.5.2. Filtro de Madgwick y cuaterniones

Existen múltiples formas de eliminar la gravedad. Los filtros de paso bajo o filtro de paso alto (que introducen retardo), filtro de Kalman, filtro Complementario o los filtros de Mahony y Madgwick, son múltiples caminos que podemos tomar para acercarnos a la solución de este problema.

Se decidió usar una forma más precisa y simple de computar, el filtro de Madgwick [68], un filtro complementario con alta precisión sin efecto significativo en el tiempo computacional, el cual nos devuelve un cuaternión que representa una orientación, fundamental para realizar la ecuación de compensación de gravedad. La idea general de Madgwick es estimar un cuaternión combinando estimaciones de posición, integrando las mediciones del giroscopio y la dirección del acelerómetro, además de utilizar el magnetómetro para saber donde se encuentra el norte magnético. El algoritmo consta de cuatro partes [69]:

1. Cálculo de la orientación a partir de las velocidades angulares medidas por el giroscopio.

2. Cálculo de las orientaciones a partir de los vectores medidos del campo gravitacional y del campo magnético.

3. Fusión de las dos estimaciones anteriores.

4. Normalización del cuaternión de la medición.

Los cuaterniones son una extensión de los números reales, como lo son los números complejos, que son un par ordenado de números reales que tienen una parte compleja y una imaginaria (la unidad imaginaria i). Se extienden más allá de estos ya que añaden las unidades imaginarias i, j y k [70].

La explicación exhaustiva del algoritmo se encuentra en el paper escrito por Sebastian O.H Madwick, el creador del mismo [71], pero para entender el proceso de compensación de gravedad utilizado en este proyecto se debe tener en cuenta cual es la utilidad de los cuaterniones.

Un cuaternión es un número complejo de cuatro dimensiones que puede ser utilizado para representar la orientación de un cuerpo rígido o un frame (marco) de coordenadas en un espacio tridimensional. Una orientación arbitraria de un frame B relativa a un frame A que puede ser lograda a través de una rotación del ángulo θ alrededor de un eje A_r definido en el frame A. Por ejemplo, un cuaternión ${}^{B}_{A}q$ describe la orientación del frame A relativa al frame B y A_r es un vector descrito en el frame A [71], [69].

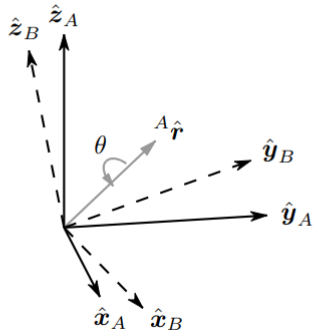


Figura 5.6: Orientación del frame B obtenido por rotación alrededor del eje A_r.

Un cuaternión conjugado se denota con $*$ y se puede usar para cambiar los frames relativos que representan la orientación. El cuaternión ${}^B_A q$, por ejemplo, tiene el conjugado ${}^A_B q$, este último describe la orientación de un frame B relativa a un frame A, que se interpreta como estar viendo el cuerpo desde otro ángulo. Por otro lado un vector tri-dimensional puede ser rotado por un cuaternión, en el siguiente ejemplo, A_v y B_v son el mismo vector descrito en el frame A y en el frame B respectivamente, tienen un 0 insertado como el primer elemento para aumentar su dimensión [72].

$$ {}^B v = {}^A_B q \otimes {}^A v \otimes {}^A_B q^* $$

Donde el símbolo $\otimes$ representa la multiplicación entre dos cuaterniones, ${}^A v$ y ${}^B v$ son las diferentes observaciones del vector v desde los dos marcos de referencia.

El algoritmo de Madgwick es usado para encontrar un cuaternión ${}^L_G q$, que representa la orientación del sensor respecto al marco de la tierra, así se logra calcular la dirección esperada de la gravedad y sustraerla de la aceleración total obtenida por el sensor. A continuación, se rota el vector de gravedad de la tierra de tres dimensiones hacia la orientación del dispositivo, es decir, por el cuaternión obtenido con Madgwick.

$$ {}^G g = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} $$

Siendo g un vector de tres dimensiones que representa la gravedad en la tierra, agregamos un elemento arbitrario 0 para representarlo como un cuaternión.

$$^{G}g = \Big(0, \quad 0, \quad 0, \quad 1\Big)$$

Luego, se utiliza la fórmula para rotar un vector de tres dimensiones por un cuaternión explicada previamente.

$$^{L}g = {}^{G}_{L}q \otimes {}^{G}g \otimes {}^{L}_{G}q$$

Al realizar los dos productos de Hamilton se obtienen los valores de los cuatro elementos del cuaternión ^{L}g. Estos deben quedar expresados como un vector de tres dimensiones para poder restarlo con el obtenido por el acelerómetro. Los tres últimos elementos del cuaternión son g_1,g_2 y g_3.

$$g_1 = 2 * (q_1 * q_3 + q_0 * q_2)$$
$$g_2 = 2 * (q_2 * q_3 + q_0 * q_1)$$
$$g_3 = q_0^2 - q_1^2 - q_2^2 + q_3^2$$

Finalmente se sustrae el vector de gravedad rotado a los datos de aceleración recopilados por el acelerómetro.

$$^{L}g = (g_1, g_2, g_3) = (g_x, g_y, g_z)$$
$$A = (a_x, a_y, a_x)$$
$$AceleraciónLineal = A - {}^{L}g$$

5.5.3. Obteniendo Aceleración.

Uno de los objetivos de este proyecto es calcular la velocidad de los deportistas durante su entrenamiento. Para esto se puede realizar una integración simple sobre la

aceleración, y después se puede volver a integrar para obtener la distancia recorrida. La siguiente integración nos devuelve la velocidad (v).

$$v = \int a * \Delta t$$

Realizando una segunda integración obtenemos la distancia recorrida:

$$x = \int v * \Delta t$$

Debido a que las mediciones del acelerómetro no son exactas e integrar produce un "drift". Esto se debe a que la mayor fuente de error es el "bias" del acelerómetro. Un bias constante de error ε doblemente integrado causa un error de posición que aumenta de manera cuadrática a través del tiempo. Se podría estimar el bias midiendo un promedio de las mediciones del acelerómetro al estar en reposo. El MPU 9250 realiza por su cuenta una corrección de bias pero el mismo no fue suficiente para evitar el crecimiento del error al integrar para obtener la posición [73], [74], [75].

```
1  MadgwickQuaternionUpdate(IMU.ax, IMU.ay, IMU.az, IMU.gx * DEG_TO_RAD,
       IMU.gy * DEG_TO_RAD, IMU.gz * DEG_TO_RAD, IMU.my, IMU.mx, IMU.mz,
       IMU.deltat);
2
3  IMU.delt_t = millis() - IMU.count;
4  float g[3];
5
6  g[0] = 2.0f * (*(getQ() + 1) * *(getQ() + 3) - *getQ()  * *(getQ() +
       2));
7  g[1] = 2.0f * (*getQ() * *(getQ() + 1) + * (getQ() + 2) * *(getQ() +
       3));
8  g[2] = *getQ() * *getQ() - *(getQ() + 1) * *(getQ() + 1) - *(getQ() +
       2) * *(getQ() + 2) + *(getQ() + 3)  * *(getQ() + 3);
9
10 movementEndCheck(g);
11
12 ax = IMU.ax - g[0];
13 ay = IMU.ay - g[1];
14 az = IMU.az - g[2];
```

En el código representado arriba se puede ver el llamado al algoritmo de Madgwick, el mismo recibe los datos de los sensores, en el caso del giroscopio en formato de radianes, y el tiempo que le tomo obtener los datos desde la última vez para poder realizar la integración necesaria en el algoritmo. Una vez obtenido el cuaternión, se realiza el cálculo explicado previamente para rotar el vector de gravedad de tres dimensiones por el cuaternión obtenido. Por último, se sustraen estos valores obtenidos de los valores en crudo del acelerómetro. Debido a que la forma de acceder a los elementos del cuaternión es a través de punteros, se puede dificultad un poco la lectura de la correspondiente línea de código. Clarificando el código sin acceso a punteros este se vería de la siguiente forma:

$$g[0] = 2 * (q[1] * q[3] - q[0] * q[2]))$$
$$g[1] = 2 * (q[0] * q[1] + q[2] * q[3])$$
$$g[2] = q[0] * q[0] - q[1] * q[1] - q[2] * q[2] + q[3] * q[3]$$

5.5.4. Verificación de movimiento

Una vez calculada la aceleración, se debe hacer una verificación de movimiento, esto se debe a que cuando un cuerpo en movimiento empieza a disminuir su velocidad, este tiene una aceleración negativa. Otra de decir esto es, cuando el signo de la aceleración es el opuesto al de velocidad, nos vemos frente a una disminución de la velocidad, en caso de que tengan el mismo signo, estamos aumentando la velocidad. Cuando realizamos la integración y obtenemos la velocidad, esta misma es sumada al valor previamente obtenido. Como este valor siempre se suma a la velocidad previa después de realizar la integración, nunca veríamos la velocidad volviendo al valor cero y pensaríamos que el dispositivo está en constante movimiento. Por esta razón, es crucial "reiniciar" el valor de la velocidad volviendo a asignar cero. Si un cierto número de muestras es igual a cero, simplemente volvemos a asignar cero en el valor acumulativo de velocidad.

```
1 if ((IMU.ax - g[0]) > 0.100 || (IMU.ax - g[0]) <  -0.100 || (IMU.ay -
    g[2]) > 0.100 || (IMU.ay - g[2]) < -0.100 || (IMU.az - g[1]) >
    0.100 || (IMU.az - g[1]) < -0.100) {
2     control = 25;
```

```
  }
  else {
    control--;
    if (control == 0) {
      control = 25;
      previous_acceleration = 0;
      previous_real_acceleration = 0;
      current_velocity = 0;
    }
  }
```

5.5.5. Reducción de ruido mecánico por software

Como se mencionó previamente, factores como el posible error del sensor, vibraciones, precisión del sensor, y el hecho de que el dispositivo está montado sobre la espalda del deportista evitan tener una lectura de aceleración = 0 aunque el deportista no esté en movimiento.

Teniendo en cuenta esto, en un escenario en el mundo real nos veríamos con un problema ya que el dispositivo raramente reportará el valor cero en todos sus ejes por más que el cuerpo parezca estar quieto. Cuando el dispositivo no se está moviendo, el error del sensor y las pequeñas vibraciones que percibe el mismo pueden ser interpretadas como velocidad constante debido a que las muestras que están siendo sumadas no son cero. Lo ideal sería que en una situación en la que se sostiene el dispositivo y no hay movimientos fuertes por parte de quien lo sostiene (correr, caminar o saltar), estos valores sean cero. En el siguiente fragmento de código realizamos el filtrado explicado, el umbral es +-0.100

```
if (ax < 0.100 && ax > -0.100 ) ax = 0;
if (ay < 0.100 && ay > -0.100 ) ay = 0;
if (az < 0.100 && az > -0.100 ) az = 0;
```

5.5.6. Integrando la aceleración para obtener el resultado final

Una vez aplicados los filtros requeridos sobre los datos en crudos, se procede al cálculo de la velocidad. Primero debemos obtener la aceleración total, dado los tres ejes de aceleración de un cuerpo, esta se calcula realizando el módulo del vector de aceleración, el cual es un número real que representa la longitud del vector:

$$a = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

```
ax *= 9.8;
ay *= 9.8;
az *= 9.8;

float linear_accel = sqrt((ax * ax) + (ay *  ay) + (az * az));
avg_accel += linear_accel;
```

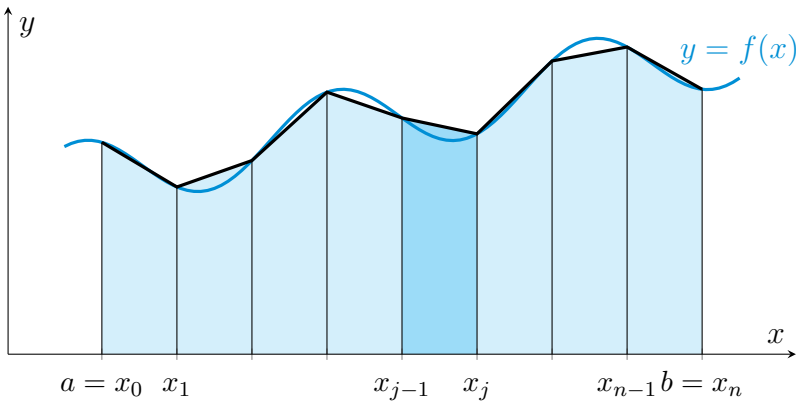
Una vez conseguida la aceleración total, se realiza la integración. Para la misma se utilizó la Regla del Trapecio, el lenguaje C++ de por si no ofrece ninguna función para calcular integrales y se consideró que agregar una librería de matemática compleja para una única función no era necesario, en especial en un dispositivo de poca memoria como Adafruit Feather Huzzah. La Regla del Trapecio, junto con la Regla del Punto medio y la Regla de Simpson, son las tres formas más simples de calcular una integral con las operaciones aritméticas que nos ofrece C++ que requieren menor poder de cómputo. Esta fórmula se vuelve más precisa a medida que la resolución de la partición aumenta, es decir un gran número de muestras y un intervalo de tiempo pequeño.

La regla del trapezoide es un método para calcular aproximadamente el valor de una integral definida. Se dice que es una "mejora" del método de las Sumas de Riemann, la cual utiliza rectángulos para calcular el área bajo la curva. Con la Regla del Trapezoide se utilizan trapecios que nos permiten obtener aproximaciones más precisas.

$$\int_a^b f(x)dx \approx \sum_{n=0}^{N-1} \frac{1}{2}(\Delta x)_n(f_n + f_{n+1})$$

$$(\Delta x)_n = x_n - x_{n-1}$$

Donde $(\Delta x)_n$ representa el intervalo en tiempo entre cada muestra.



Para realizar una reducción de ruido en la toma de datos se realizó un promedio de las muestras. Se tuvieron en cuenta dos posibles soluciones. Una de ellas era tomar muestras durante un tiempo determinado, como por ejemplo 30ms, y promediar la cantidad de muestras tomadas en ese tiempo. La otra posible solución era tomar un número de muestras fijas y calcular cuanto tiempo se tardó en tomar las mismas. Debido a los posibles retardos y/o errores del sensor, se optó por evitar la primera solución para asegurarnos de que el promedio obtenido siempre corresponda a una cantidad constante de muestras. A continuación se muestra un fragmento de código de la solución implementada:

```
1  if (count_messages == 30) {
2
3    avg_delta = millis() - count_delta;
4    avg_accel /= 30;
5    count_delta = millis();
```

```
6
7       current_acceleration = avg_accel - previous_acceleration;
8
9       current_velocity += ((current_acceleration +
        previous_real_acceleration) * 0.5) * (avg_delta / 1000.0);
10
11      previous_velocity = current_velocity;
12      previous_real_acceleration = current_acceleration;
13      previous_acceleration = avg_accel;
14
15      count_messages = 0;
16
17      String msg;
18      msg += device_id + "," + String(current_acceleration , 6) + "," +
        String((current_velocity)* 3.6, 6);
19
20      sendTopicData("/imu", msg);
21     }
22 }
```

5.5.7. Arreglo en re-conexión y problema de datos basura

Uno de los problemas encontrados durante el desarrollo del dispositivo al broker MQTT. Debido al gran envío de datos que realizar por segundo y la cantidad de dispositivos en la red, es posible que el broker MQTT pueda desconectarse y reconectarse. Como se vio previamente, el MPU y el GPS pueden recopilar decenas de muestras por segundo, enviar esa cantidad de muestras por WiFi no solo saturaría la red, sino que no sería necesario por la cantidad de movimiento que se puede realizar con el dispositivo en ese tiempo. Para enviar los datos, se decidió obtener un promedio de los mismos y enviarlos al broker. El dispositivo envía cada 30 muestras procesadas del MPU, los datos al broker MQTT, lo cual es aproximadamente 3 muestras por segundo, suficientes para mostrar datos en tiempo real y evitar saturar la red. Sin embargo, es posible que el broker sufra algún tipo de caída, ya sea por problemas en el host que lo ejecuta, o desconexiones en la red WiFi.

El ESP8266 es un dispositivo single-core que no es capaz de paralelizar, es decir, todas las acciones que ejecutamos en nuestro loop principal se ejecutan secuencial-

mente, y ninguna otra acción se puede ejecutar en el mismo instante de tiempo, es capaz de soportar concurrencia, pero no paralelismo puro. Cuando ocurre una caída de conexión, el dispositivo debe dejar de procesar datos e intentar reconectarse al broker hasta lograrlo exitosamente. Por otro lado, el dispositivo carece de un reloj interno, y en caso de tenerlo, debería estar alimentado por una batería constantemente. Esta situación representa un problema al realizar la integración de los datos.

Para realizar la integración que permite obtener la velocidad, se debe calcular el tiempo que se tarda en tomar las muestras, como el dispositivo carece de un reloj interno, la forma de tomar una referencia del tiempo es en base a la función $millis()$. La definición de millis muestra como la misma retorna el número de milisegundos desde que la placa empezó a correr el programa que tiene grabado,el mismo regresa a 0 después de aproximadamente 50 días en funcionamiento. Para realizar la integración, se toma el tiempo al empezar a tomar la primera muestra con millis, cuando se toman las 30 muestras, se vuelve a utilizar la función millis. La diferencia entre el tiempo al iniciar a tomar las muestras y el tiempo al tomar la última muestra es el tiempo transcurrido. En caso de una desconexión con el broker MQTT en el proceso de muestreo, el dispositivo intentará reconectarse y no seguirá tomando muestras en tiempo real hasta que se re-conecte. Para solucionar el problema, se agregó la función $fallbackFunction()$ la cual es llamada cuando el dispositivo se desconecta del broker MQTT.

```
void fallbackFunction(){
    current_acceleration = 0;
    current_velocity = 0;
    previous_velocity = 0;
    previous_real_acceleration = 0;
    previous_acceleration = 0;
    count_messages = 0;
    count_delta = millis();
}
```

Es fundamental reiniciar la toma de muestras y volver a tomar el tiempo cuando el dispositivo se re-conecta, para evitar que los datos de la integración sean alterados con valores inválidos. Si esto no se hace, el dispositivo acumulará valores erróneos hasta que el mismo reinicie sus valores tras dejar de moverse por un pequeño lapso

de tiempo. Otra posible solución sería la paralelización de las tareas, para que el dispositivo pueda conectarse y seguir tomando muestras que serán enviadas tan pronto como el mismo se re-conecte, la misma queda fuera del alcance de esta tesina.

5.5.8. Obteniendo la distancia recorrida

Para calcular la distancia total recorrida se decidió utilizar el GPS como fuente de información, como se mencionó previamente en este capítulo, el IMU no es una fuente confiable para obtener el desplazamiento. Con el GPS, es posible obtener la distancia en metros que existe entre dos posiciones representadas con coordenadas (latitud y longitud). La función distanceBetween( ) de TinyGPS++ se encarga de hacer este cálculo utilizando la Great-Circle Distance (distancia ortodrómica), es decir la distancia del camino más corto entre dos puntos de la superficie terrestre, que se computa utilizando un radio hipotético de 6372795 metros de la tierra. El mismo aclara que como la Tierra no es una esfera perfecta, puede haber errores de hasta un 0.5 %, que en el caso de este dispositivo son irrelevantes ya que las distancias que se calculan no llegan a ser afectadas por la curvatura de la Tierra. En el siguiente código se puede ver la función getGPSReadings() que obtiene el fix más reciente y calcula la distancia recorrida desde el último fix, además de almacenar la ultima posición para poder hacer el cálculo de distancia en el siguiente ciclo. El primer cálculo de distancia es incorrecto ya que no se tiene una posición anterior (las variables son inicializadas en longitud y latitud 0), pero este mismo es descartado por el broker MQTT. Además, desde esta función se envían las coordenadas y la velocidad.

```
1  void getGPSReadings(unsigned long ms)
2  {
3    unsigned long start = millis();
4    do
5    {
6      while (SSerial.available())
7      {
8        neoM8N.encode(SSerial.read());
9      }
10   }
```

```
11   while (millis() - start < ms);
12
13   if(neoM8N.location.isUpdated())
14   {
15    String msg;
16    double  distance = neoM8N.distanceBetween(neoM8N.location.lat(),
      neoM8N.location.lng(),last_lat , last_lng);
17    msg += device_id +  "," + String(neoM8N.location.lat(),14) + ","+
      String(neoM8N.location.lng(),14)+","+String(neoM8N.speed.kmph(),7)+
      ","+String(distance);
18    last_lng = neoM8N.location.lng();
19    last_lat = neoM8N.location.lat();
20
21    sendTopicData("/gps", msg);
22   }
23 }
```

Cada vez que el dispositivo obtiene un nuevo Fix, se calcula la distancia entre la última posición obtenida y la nueva, para luego ser totalizado. El dato es enviado junto con los valores de ubicación más recientes. Aun así, este método puede acarrear errores ya que dependen de la capacidad de polling del sensor y la señal.

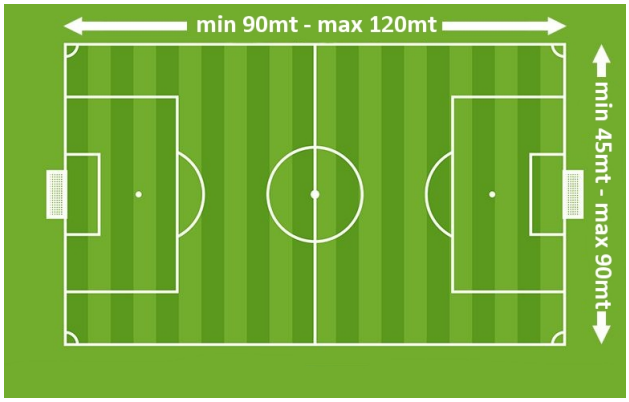


Figura 5.7: Tamaño reglamentario de una cancha de Fútbol profesional establecido por la Fifa.

Capítulo 6

Pruebas del prototipo y comparativas

6.1. Recibiendo y Procesando la data

6.1.1. Mosquitto broker MQTT

Como se ha mencionado en el marco teórico, MQTT es un protocolo de conexión que explica cómo los bytes son ordenados y enviados por la red TCP/IP. Todas las bibliotecas y herramientas de MQTT ofrecen distintas formas sencillas de manejar los mensajes del protocolo, como por ejemplo, los mensajes CONNECT y SUBSCRIBE. En este caso, se estará utilizando Mosquitto como broker MQTT, pero existen otros como VerneMQ, EMQ, HiveMQ, entre otros. Mosquitto fue creado por la Eclipse Foundation, es de código libre e implementa las versiones 5.0, 3.1.1 y 3.1 de MQTT mediante el modelo publish/subscribe, y es ampliamente utilizado en el mundo IoT ya que funciona posee una muy buena performance en dispositivos de baja consumo [76], [77], [78].

Para comenzar, se describirá el mensaje CONNECT, utilizado para establecer una conexión entre el cliente y el broker. El mensaje CONNECT contiene algunos de los siguientes parámetros:

- **cleanSession**: indica si la conexión debe ser persistente o no. Si se indica que sea persistente, almacena todas las suscripciones y los mensajes posibles perdidos (dependiendo del QoS) en el broker.
- **lastWillTopic**: si la conexión se cae de manera inesperada, el broker automáticamente publicará en un topic un mensaje de "ultimo deseo".
- **lastWillQoS**: indica la QoS del mensaje de "último deseo".
- **lastWillMessage**: el mensaje en sí de "último deseo".
- **keepAlive**: indica el intervalo en que el cliente necesita realizar un ping en el broker para mantener la conexión activa.

Cuando el broker recibe el mensaje CONNECT, envía un mensaje CONNACK hacia el cliente. Este mensaje contiene los siguientes parámetros:

- **sessionPresent**: indica si la conexión ya contiene una sesión persistente. Si la conexión ya contiene topics suscritos, recibirá la entrega de los mensajes ausentes.
- **returnCode**: 0 indica éxito, otros valores indican alguna falla.

Una vez que el cliente recibe el CONNACK la conexión queda establecida y podrá enviar mensajes a través de diferentes topics. Para el desarrollo de este proyecto, los clientes son los dispositivos de hardware que enviarán mediante distintos topics la aceleración y velocidad proviniendo del IMU, y la latitud, longitud, distancia y velocidad proveniente del GPS.

El broker también puede recibir mensajes PUBLISH y SUBSCRIBE por parte de los clientes. El mensaje SUBSCRIBE lo envían los clientes hacia el broker para subscribirse a los topics de interés, recibiendo los mensajes que se publiquen en ese topic por parte de otros clientes. Algunos de los parámetros de SUBSCRIBE son:

- **qos** : indica con qué consistencia los mensajes en un determinado topic necesitan ser entregados a los clientes, siendo como ya hemos visto, 0 no confiable,

mensaje entregado como máximo una sola vez y si el cliente no se encuentra disponible se perderá el mensaje; 1 donde el mensaje debe entregarse al menos una vez; 2 el mensaje debe entregarse exactamente una vez.

- **topic**: canal al cual suscribirse para recibir los mensajes que se publiquen en él.

Luego de que el cliente se haya suscrito a un topic exitosamente, el broker le enviará un mensaje SUBACK con uno o más parámetros returnCode, siendo estos valores de 0 a 2 mensajes de éxito como el nivel de QoS elegido, o valor 128 donde informa un fallo. El cliente también puede enviar el mensaje UNSUBSCRIBE de uno o más topics.

El mensaje PUBLISH es utilizado para enviar mensajes al broker. Este mensaje está formado por un topic y un payload (o carga útil). Luego, el broker puede enviar este mensaje a todos los clientes que decidan suscribirse a este topic. PUBLISH posee los siguientes parámetros:

- **topicName**: el tema en el cual el mensaje es publicado.
- **qos**: nivel de la calidad de servicio de la entrega del mensaje.
- **payload**: el mensaje en sí mismo por lo general en formato String.

Figura 6.1: Nueva conexion al servidor de Mosquitto MQTT.

En esta imagen se ve como mediante el cliente de Mosquitto realiza la suscripción al broker a través del topic *device*1/*imu* a través del mensaje SUBSCRIBE y respondiendo con el mensaje SUBACK. De esta manera, el cliente Mosquitto queda suscrito correctamente quedando a la espera de que se publiquen mensajes en ese topic.

```
mosquitto_pub -h 192.168.0.7 -p 1883 -t "/imu" -m "device1,1.1,10.0"
mosquitto_sub -h 192.168.0.7 -p 1883 -t "/imu" -v
```

Las anteriores instrucciones del cliente Mosquitto pueden utilizarse para publicar mensajes de ejemplo hacia el topic */imu* como también para suscribirse y recibir los mensajes publicados en ese topic, siendo en este caso, mediante *mosquitto_pub* enviando $acceleration = 1{,}1$ y $velocity = 10{,}0$, y recibiéndolo mediante la instrucción *mosquitto_sub*. Más adelante se mostrará este funcionamiento con los datos reales enviados por el dispositivo y recibidos por el broker en tiempo real.

Por último, pueden definirse algunas configuraciones en el archivo de configuración de Mosquitto en "etc/mosquitto/mosquitto.conf":

```
persistence false
persistence_location /var/lib/mosquitto/
port 1883
log_dest file /var/log/mosquitto/mosquitto.log
```

- **persistence false**: ya que no se desea persistir ningún mensaje que reciba el broker debido a que toda esta información se almacenará en InfluxDB, aunque también puede configurarse con el valor true y la información se persistirá en "var/lib/mosquitto".
- **port 1883**: puerto por defecto del protocolo MQTT en el cual correrá Mosquitto.
- **log_dest file**: tipo de salida y ubicación donde pueden verse los logs de Mosquitto (conexiones, desconexiones, errores, versiones, sockets escuchando en determinado puerto, path donde la database de Mosquitto donde se persiste la información, archivo de configuración utilizado, etc).

6.1.2. Leyendo y procesando los datos del broker

Los dispositivos envían los datos son manipulados mediante filtros y algoritmos, luego de recibir esos datos en el broker, se necesita una forma de persistir esa

información en una base de datos, debido a que el broker solamente los almacena de forma temporal en un buffer en memoria RAM, es decir, en memoria volátil.

Para lograr esto y obtener los datos que son recibidos por el broker, se creó un script en Python que se suscribirá a distintos topics con el fin de recibir los datos que los dispositivos enviaron mediante mensajes publish y, a medida que los datos son leídos de los topics, procesarlos y formatearlos para ser almacenados en InfluxDB en un formato que este permita, y a su vez con una estructura definida que favorezca la performance de las consultas por parte de Grafana para visualizar las métricas en tiempo real.

A continuación se explicará parte por parte el código escrito en Python. Para comenzar, se decidió utilizar las librerías *pahomqtt*, *influxdb* y *timeit*. Esta última es una librería que ya trae incorporada Python junto con su instalación, pero *pahomqtt* e *influxdb* pueden ser instaladas de la siguiente manera [79], [80].

```
pip install paho-mqtt
pip install influxdb
```

Las librerías se importarán en el script llamado subs-paho-mqtt.py.

```
import ssl
import sys

from influxdb import InfluxDBClient
from paho.mqtt.client import mqtt
from timeit import default_timer as timer
```

Se comenzará explicando la función main() donde se encuentran las funciones principales para conectarse al broker MQTT y a la base de datos InfluxDB.

```
def main():
    global influx_client
    global sprint_counter
    global sprint_duration

```

```
6       sprint_counter = False
7       sprint_duration = 0.0
8
9       influx_client = InfluxDBClient('localhost', port=8086, database='
        tesis', username='root', password='root')
10      influx_client.switch_database('tesis')
11      client = mqtt.Client(client_id='mqtt-subscriber', clean_session=
        False, userdata=None, transport="tcp")
12      client.on_connect = on_connect
13      client.on_message = on_message
14      client.connect(host='192.168.0.7', port=1883, keepalive=60)
15      client.loop_forever()
16
17  if __name__ == '__main__':
18      main()
```

En este código se define el objeto *influx_client* en Python, que se utilizará para interactuar con InfluxDB. Se comienza asignando la información necesaria para que el cliente se conecte a nuestra base de datos, entre ellos el host, puerto, nombre de base de datos y la credenciales para poder manejarla. Luego, se crea una variable *client* que utilizará la libreria *pahomqtt* y servirá como cliente para acceder al broker MQTT. Los parámetros que se definen a la hora de instanciar el cliente son: *client_id*, el nombre que tendrá este cliente que se conectará el broker, $clean\_session = False$, para indicar que el cliente es un cliente persistente y la información de la suscripción al broker deben retenerse al igual que los mensajes encolados cuando el cliente se desconecte. Si este parámetro fuese True, el broker removería toda la información sobre el cliente cuando este se desconecte; $transport =' tcp'$, para indicar que se utilizará *raw tcp* como capa de transporte en lugar de $WebSockets$. Además, se inicializan las variables globales *sprint_counter* y *sprint_duration* que se utilizarán para contar los sprints y la duración de cada uno de estos a lo largo de la ejecución del script.

Una vez instanciado el cliente MQTT, se procede a asignar los callbacks correspondientes a *on_connect* y *on_mesagge*, las cuales se encargan de leer los mensajes de los topics del broker y realizar tareas según el topic leído respectivamente. Siguiente a esto se realiza la conexion broker utilizando los parámetros: *host*, será la dirección IP del broker, para el desarrollo de este trabajo será la dirección IP que se definirá estáticamente en una Raspberry Pi 4, la cual ejecutará el broker, InfluxDB y Gra-

fana automáticamente al encenderse; *port* = 1883, el puerto por defecto utilizado por el protocolo MQTT; *keepalive* = 60, el período de tiempo máximo en segundos para la comunicación entre el cliente y el broker. Este parámetro determina el intervalo de tiempo en que el cliente envía PING's al broker en caso de que cese el intercambio de mensajes, es decir, que el cliente por alguna razón no envíe más información ni del IMU ni del GPS. Mediante los mensajes PING se mantendrá la conexión establecida hasta que el dispositivo pueda recuperarse y volver a enviar información de los sensores.

Finalmente se llama a *client.loop_forever*(), esta función internamente llama a la función *loop* de la librería paho-mqtt que hará que el programa se ejecute infinitamente y a su vez maneje las reconecciones automáticamente en caso de que sea necesario. Se puede decir que funciona de la misma manera que el loop() que se vió previamente al explicar el código C++ del dispositivo.

Como se ha mencionado previamente, las funciones *on_connect* y *on_mesagge* son callbacks que se ejecutan cuando el cliente detecta que ocurrieron estas acciones. En el caso de *on_connect*, la misma esta definida por:

```
def on_connect(client, flags, rc):
    print('connected (%s)' % client._client_id)

    client.message_callback_add("/gps", on_message_gps)
    client.message_callback_add("/imu", on_message_imu)
    client.message_callback_add("/logs", on_message_logs)

    client.subscribe("#", 0)
```

Esta función es la responsable de suscribirse a los distintos topics de interés en el broker donde los dispositivos envían constantemente información. Esto se logra mediante el llamado a las funciones del cliente MQTT *message_callback_add*(). Estas funciones leen los mensajes publicados en los diferentes topics, y mediante ellas el cliente se subscribe a múltiples topics simultáneamente, que de otra forma no sería posible, entonces al leer información de un determinado topic, se lanza un llamado a las funciones callback que se encargarán del manejo de los datos. Por lo tanto, al leer del topic */gps* se llama a la función *on_message_gps*, al leer */imu* se llama a *on_message_imu* y al leer */logs* se llama a *on_message_logs*. De esta forma,

todos los devices conectados enviarán sus datos a estos topics donde se distribuirán en los métodos callbacks para hacer los cálculos correspondientes.

Por último se llama a la función suscribe, que permite ejecutar las anteriores funciones callback para realizar la suscripción a cada topic con QoS 0, comenzando la búsqueda de los topics por el root level o #. A continuación se muestra el código de las funciones *on_message*, *on_message_gps*, *on_message_imu* y *on_message_logs*.

```
def on_message(client, userdata, message):
    print('------------------------------')
    print('Wrong topic, please send the data to the correct topic')
```

Esta función será llamada desde la función main() en caso de que el dispositivo envíe información a un topic erróneo, es decir, donde este cliente (o script) no esté suscrito a uno de esos topics en el que el cliente haya enviado datos. En caso de que esta función sea llamada, se imprimirá en la terminal donde se haya ejecutado el cliente el mensaje de error "Wrong topic, please send the data to the correct topic", para informar que el dispositivo está enviando información a un topic donde no se están leyendo los datos. Por lo tanto, quien define la nomenclatura de los topics, es el cliente Python quien se suscribe al broker y no los dispositivos quienes publican la información.

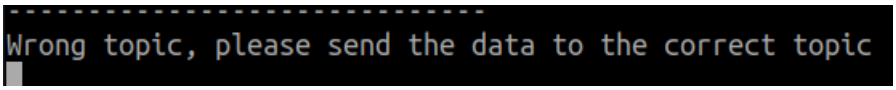


Figura 6.2: Lectura de datos en el broker desde topics incorrectos.

```
    def on_message_logs(client, message):
        message.payload = str(message.payload, 'utf-8')
        message.payload = message.payload.split(',')

        try:
            device_id = message.payload[0]
            log = message.payload[1]

            print('------------------------------')
            print('device: %s' %device_id)
            print('topic: %s' %message.topic)
            print('payload: log: %s' %log)
```

```
13              print('qos: %d' % message.qos)
14
15              json_body = [{
16                  "measurement": device_id,
17                  "tags": {
18                      "topic": "logs"
19                  },
20                  "fields": {
21                      "log": log
22                  }
23              }]
24
25              influx_client.write_points(json_body, database='tesis')
26
27          except Exception:
28              pass
```

Esta función se llama desde la función que se explicó anteriormente *on_connect* y recibe constantemente los mensajes que los dispositivos publican en el topic de logs, como por ejemplo, información de cuando el dispositivo busca contarse a una red WiFi, cuando conecta al servidor MQTT, errores, y toda información que sea de utilidad que se verá reflejada en la terminal durante la ejecución del cliente y que a su vez podrá verse en un panel específico de Grafana.

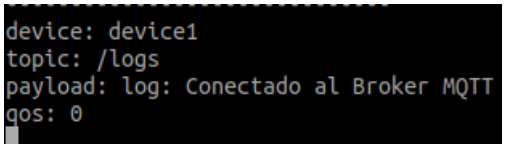


Figura 6.3: Conexión al topic /logs en el broker.

Cabe destacar, que el usuario final no tiene por qué ver toda esta información que corre en segundo plano, solo es de utilidad para los desarrolladores o personal técnico que desee saber con más detalle el flujo de ejecución del programa en su totalidad o en caso de que se necesite debuggear ante la presencia de errores inesperados. A su vez, en esta función se está creando un objeto JSON que formateará los datos leídos en un formato entendible por InfluxDB, donde se declara el nombre del *measurement* (tabla en una base de datos relacional, tal cual se vió en el marco teórico) llamado con el nombre asignado al device conectado que envía la data donde

se almacenarán los logs en la base de datos. En el mismo objeto JSON definimos dentro del atributo *tags*, un nuevo objeto compuesto por la clave topic y como valor *logs* (que como se ha explicado, es necesario que exista al menos un *tag* que servirá para indexar la información más rápidamente en las futuras consultas), y otro objeto *fields* compuesto por la clave *log* y valor la variable log que contendrá el mensaje publicado por el dispositivo. Esta es la estructura necesaria que determina InfluxDB a la hora de almacenar nuevos datos, consta de definir un measurement (tabla), como mínimo un *tag*, y los *fields* que se desean almacenar, que en este caso es la información enviada por los dispositivos.

Por último, se guarda el objeto JSON en InfluxDB utilizando el cliente previamente definido con la conexión ya establecida.

```
def on_message_gps(client, message):
    message.payload = str(message.payload, 'utf-8')
    message.payload = message.payload.split(',')

    try:
        device_id = message.payload[0]
        latitude = float(message.payload[1])
        longitude = float(message.payload[2])
        speed = float(message.payload[3])
        distance = float(message.payload[4])

        print('-------------------------------')
        print('device: %s' % device_id)
        print('topic: %s' % message.topic)
        print('payload: latitude: %s longitude: %s speed: %s distance:
    %s' %(latitude, longitude, speed, distance))
        print('qos: %d' % message.qos)

        if 0.0 <= speed <= 35.0:
            json_body = [{
                "measurement": device_id,
                "tags": {
                    "topic": "gps"
                },
                "fields": {
                    "latitude": latitude,
                    "longitude": longitude,
                    "speed": speed,
```

```
                    "distance": distance
                }
            }]

            influx_client.write_points(json_body, database='tesis')

    except Exception:
        pass


def on_message_imu(client, message):
    message.payload = str(message.payload, 'utf-8')
    message.payload = message.payload.split(',')

    try:
        device_id = message.payload[0]
        current_acceleration = float(message.payload[1])
        current_velocity = float(message.payload[2])

        print('-------------------------------')
        print('device: %s' % device_id)
        print('topic: %s' % message.topic)
        print('payload: current_acceleration %s current_velocity: %s'
    % (current_acceleration, current_velocity))
        print('qos: %d' % message.qos)

        if 0.0 <= current_velocity <= 35.0:
            json_body = [{
                "measurement": device_id,
                "tags": {
                    "topic": "imu"
                },
                "fields": {
                    "current_acceleration": current_acceleration,
                    "current_velocity": current_velocity
                }
            }]

            if current_velocity >= 18.0:
                if not sprint_counter:
                    sprint_counter = True
                    sprint_duration = timer()
            elif sprint_counter and current_velocity < 14.0:
```

```
70                  sprint_counter = False
71                  sprint_duration = round((timer() - sprint_duration) /
    60, 2)
72
73                  json_body["fields"].update(
74                      {
75                          "cant_sprints": 1,
76                          "sprints_duration": sprint_duration
77                      }
78                  )
79
80                  sprint_duration = 0.0
81
82              influx_client.write_points(json_body, database='tesis')
83
84      except Exception:
85          pass
```

Es necesario remarcar que se ha limitado la velocidad recibida por el dispositivo debido a que las partes que lo componen no son de la mejor calidad y puede que ocasionalmente calcule resultados inusuales. A su vez, debido a la naturaleza del algoritmo, los valores son integrados y el error se acumula. Es por este motivo y para evitar cualquier tipo de eventualidad que se decidió limitar el rango de valores de las muestras. Como en la velocidad que el dispositivo mide no se indica un vector de dirección (ya que no es necesario), nunca se podrá tener una velocidad negativa, por lo tanto los valores esperados por el broker MQTT siempre serán mayor y/o iguales a 0km/hs. Como cota superior, se utilizó 35km/hs, debido a que la velocidad máxima a la cual un ser humano puede llegar corriendo es 45km/hs en carreras cortas de 100 metros (record establecido por Usain Bolt). En [81] se puede ver que la velocidad máxima a la cual los jugadores de fútbol llegan a corren en la cancha es aproximadamente 33km/hs en caso de jugadores profesionales [82], [83], [84].

Las funciones *on_message_gps* y *on_message_imu* son muy similares a la función *on_message_logs*, donde el comportamiento es prácticamente el mismo: recibir los mensajes que fueron publicados en un determinado topic, donde en estos casos también deben convertirse los mensajes que llegan en formato String a número flotante para ser tratados correctamente por InfluxDB, imprimir la data en la terminal en ca-

so de que se desee debuggear, crear un objeto JSON con la estructura necesaria para almacenar la información en la base de datos (declarando el atributo *measurement*, *tags* y *fields* cada uno con sus respectivos valores) y por último persistiendo la información procesada. En la funcion *on_message_imu* además se calcula la cantidad de sprints y la duración de los mismos utilizando el clock del procesador. Se considera un sprint si el dispositivo envía una velocidad mayor a 18km/hs y a su vez se mantiene por encima de los 14km/hs. Debido a que el dispositivo no siempre es demasiado preciso se tomó 14km/hs como cota inferior para considerar que el jugador aún sigue en sprint. Mientras el jugador se mantenga en esa "zona"de velocidad, se contará con el clock del procesador el tiempo que transcurra hasta salir del sprint. Una vez el jugador salga de la zona de sprint, se actualizará la variable *json_body* para persistir estos nuevos datos. Nuevamente cabe recordar, que todo este proceso y el flujo se datos se da en tiempo real.

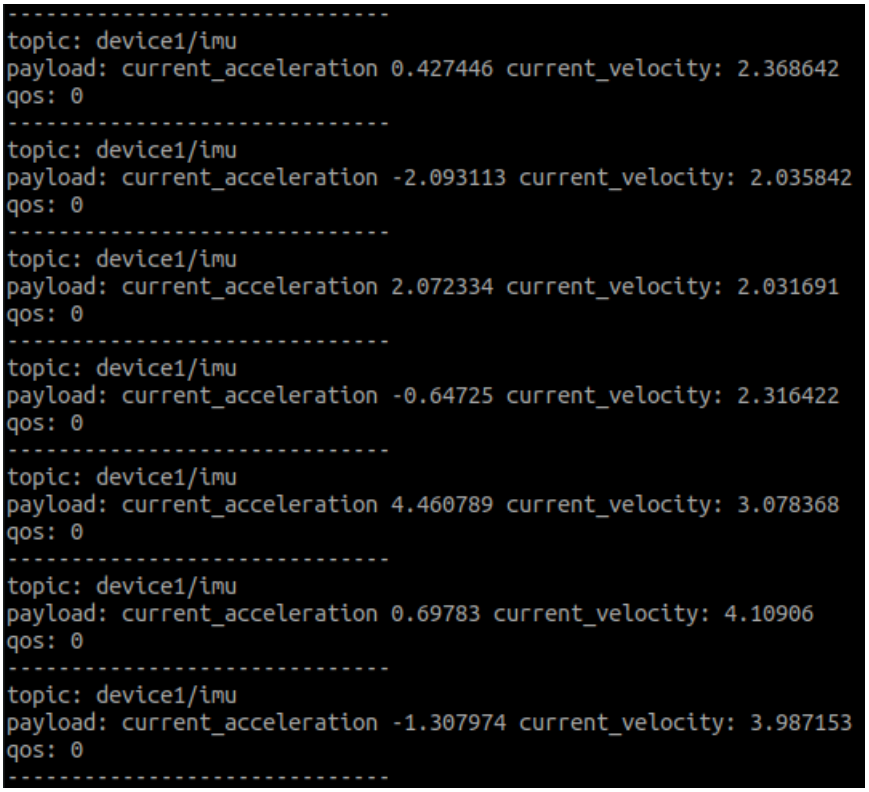


Figura 6.4: Lectura de los datos del topic device1/imu.

6.1.3. Información persistida en InfluxDB

A continuación se muestra cómo se consigue que la información sea persistida correctamente en tiempo real en InfluxDB pasando por el proceso anteriormente explicado.

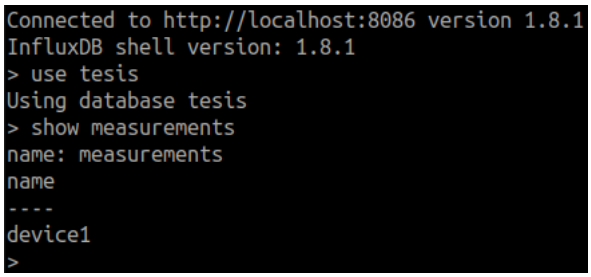


Figura 6.5: Measurements en InfluxDB.

En la foto puede verse el *measurement device*1 que fue el dispositivo que se conectó al broker y ha estado enviando los datos como se mostró anteriormente.

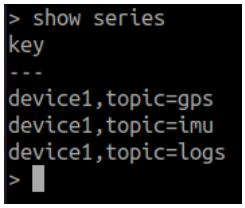


Figura 6.6: Resultado del comando show series.

En la figura 6.6 puede verse las *series* que se fueron creando al almacenar en el measurement *device*1 distintos topics desde el cliente MQTT. Se puede ver que para el measurement *device*1 se tiene el tag *topic* con valor *gps*, *imu* y *log*. Los tags que fueron definidos con el nombre topic, serán la forma de referenciar la información almacenada en InfluxDB a través de las queries de Grafana.

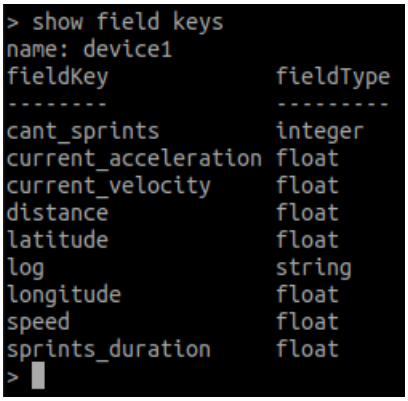


Figura 6.7: Resultado del comando show field keys.

En esta imagen pueden verse los distintos *field keys* del measurement *device*1. En este se almacenan cuatro field keys que son *latitude*, *longitude*, *speed_km* y *distance* en formato float y provenientes del topic */gps*. También se almacenan *current_acceleration*, *current_velocity*, *cant_sprints* y *sprints_duration* en formato float y provenientes del topic */imu*. Por último se almacena el field log en formato string donde se persisten los mensajes enviados por el topic */logs*. Como se ha visto, este fue el formato que se definió en el cliente MQTT con Python a la hora de definir los objetos JSON que serían guardados en InfluxDB.

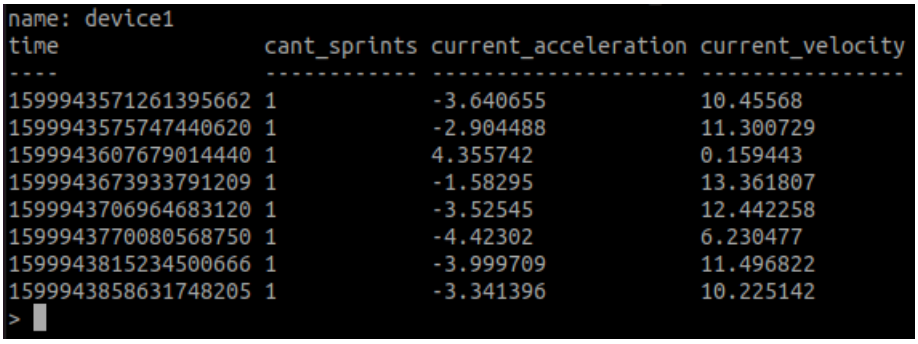


Figura 6.8: Valores almacenados para los field keys de aceleración, velocidad y sprints.

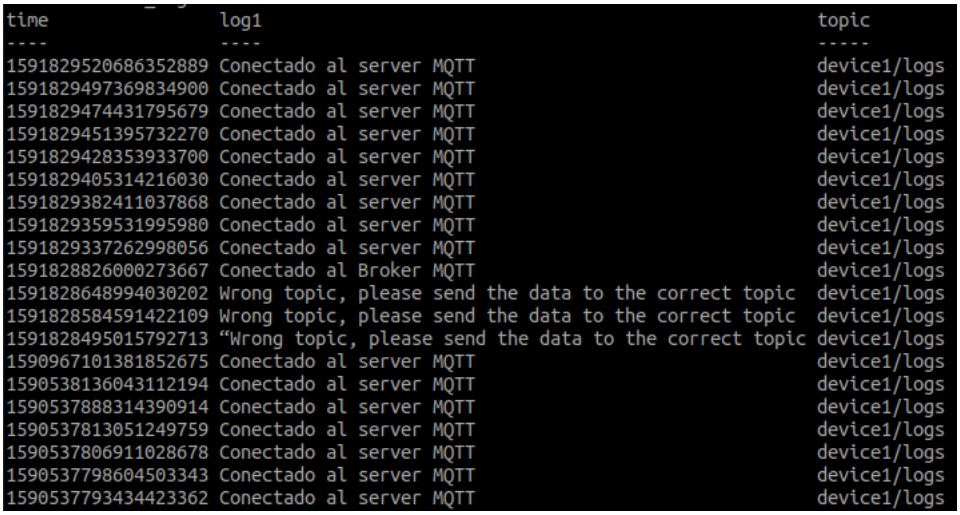
En esta imagen se puede ver cómo son almacenados en los *fields keys current_acceleration*, *current_velocity* y *cant_sprints* del topic */imu* donde se ven todos los valores que fueron enviados por el dispositivo junto con el timestamp, que es el instante de tiempo agregado automáticamente al persistir cualquier tipo de dato en InfluxDB.

De la misma forma se puede ver los datos almacenados para el topic */gps*, donde se almacenan las coordenadas que fueron emitidas por el sensor GPS en los *field keys latitude*, *longitude* y *speed_km* junto con el timestamp para cada fila.

time	latitude	longitude	speed_km	topic
1590967444241873426	-34.9159835	-57.9569365	0.5556	device1/gps
1590967444130606082	-34.9159835	-57.9569365	0.5556	device1/gps
1590967443241419286	-34.9159883	-57.956942	0.33336	device1/gps
1590967443126133337	-34.9159883	-57.956942	0.33336	device1/gps
1590967442236920612	-34.9159655	-57.9569555	0.85192	device1/gps
1590967442131647439	-34.9159655	-57.9569555	0.85192	device1/gps
1590967441307847959	-34.9159695	-57.9569575	0.85192	device1/gps
1590967441140351557	-34.9159695	-57.9569575	0.85192	device1/gps
1590967440246273323	-34.9159692	-57.9569625	1.24084	device1/gps
1590967440132507599	-34.9159692	-57.9569625	1.24084	device1/gps
1590967439242413825	-34.9159858	-57.956955	1.35196	device1/gps
1590967439127278487	-34.9159858	-57.956955	1.35196	device1/gps
1590967438244395974	-34.9159837	-57.9569583	1.18528	device1/gps
1590967438136340436	-34.9159837	-57.9569583	1.18528	device1/gps
1590967437249759048	-34.9159828	-57.9569522	0.463	device1/gps
1590967437135156586	-34.9159828	-57.9569522	0.463	device1/gps
1590967436246589513	-34.9159902	-57.9569548	0.51856	device1/gps
1590967436124286181	-34.9159902	-57.9569548	0.51856	device1/gps

Figura 6.9: Valores almacenados para los field keys correspondientes al GPS.

Por último, se muestra también cómo se almacenan los logs en la base de datos. Esta imagen es a modo de ejemplo, donde configuramos el dispositivo para que conecte y desconecte del broker MQTT y a su vez envíe información por topics donde el cliente Python suscripto no esté escuchando con el objetivo de mostrar cómo se persisten los logs enviados por el dispositivo.



time	log1	topic
1591829520686352889	Conectado al server MQTT	device1/logs
1591829497369834900	Conectado al server MQTT	device1/logs
1591829474431795679	Conectado al server MQTT	device1/logs
1591829451395732270	Conectado al server MQTT	device1/logs
1591829428353933700	Conectado al server MQTT	device1/logs
1591829405314216030	Conectado al server MQTT	device1/logs
1591829382411037868	Conectado al server MQTT	device1/logs
1591829359531995980	Conectado al server MQTT	device1/logs
1591829337262998056	Conectado al server MQTT	device1/logs
1591828826000273667	Conectado al Broker MQTT	device1/logs
1591828648994030202	Wrong topic, please send the data to the correct topic	device1/logs
1591828584591422109	Wrong topic, please send the data to the correct topic	device1/logs
1591828495015792713	“Wrong topic, please send the data to the correct topic	device1/logs
1590967101381852675	Conectado al server MQTT	device1/logs
1590538136043112194	Conectado al server MQTT	device1/logs
1590537888314390914	Conectado al server MQTT	device1/logs
1590537813051249759	Conectado al server MQTT	device1/logs
1590537806911028678	Conectado al server MQTT	device1/logs
1590537798604503343	Conectado al server MQTT	device1/logs
1590537793434423362	Conectado al server MQTT	device1/logs

Figura 6.10: Almacenamiento de Logs en InfluxDB.

6.1.4. Obteniendo y visualizando las métricas con Grafana

En este capítulo se muestra cómo se obtienen métricas reales gracias a la ejecución de consultas específicas que se definieron donde se consumen los datos almacenados en InfluxDB previamente enviados por los dispositivos. Las métricas pueden visualizarse mediante la ejecución del sistema en tiempo real, o ni bien puede seleccionarse un rango de tiempo donde ya hay información almacenada para que las consultas recaigan sobre esos datos en ese instante de tiempo. A su vez, las métricas conseguidas pueden ser exportadas en todo momento a formato .csv para que cualquier deportista u entrenador pueda visualizar las métricas en una planilla de cálculo dándole la oportunidad de poder hacer con ellas lo que le resulte de interés.

Por comenzar, se necesita realizar la conexión de Grafana con la base de datos InfluxDB, que se lleva a cabo de la siguiente manera:

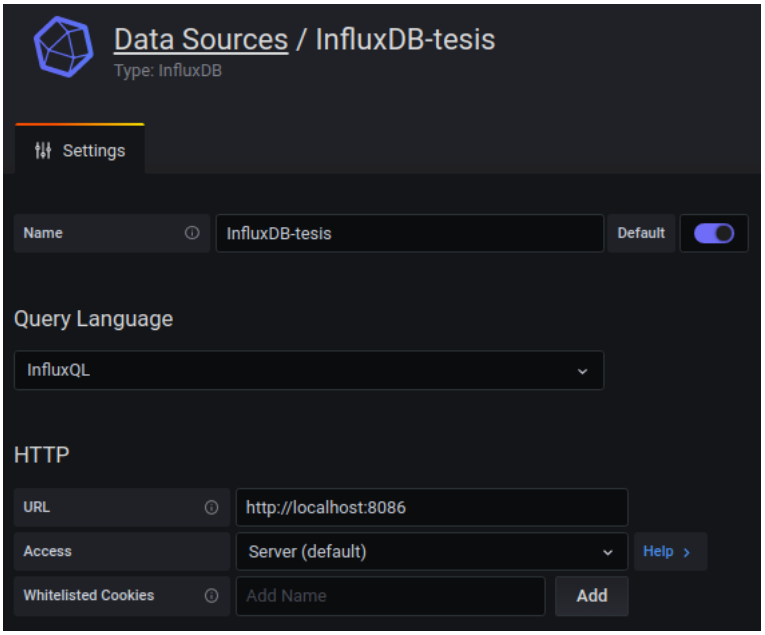


Figura 6.11: Configuración de la conexión entre Grafana e InfluxDB.

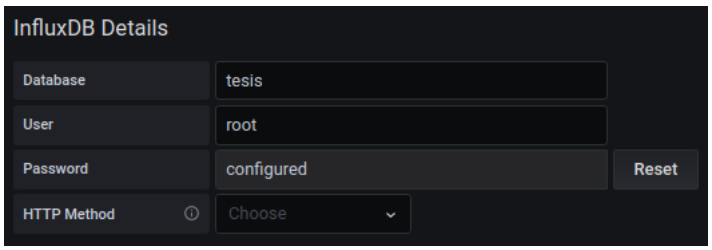


Figura 6.12: Configuración de credenciales para acceder a InfluxDB desde Grafana.

En esta configuración se define el nombre de la conexión con la base de datos que en este caso será $InfluxDB - tesis$, el lenguaje de consulta InfluxQL que es el lenguaje utilizado por InfluxDB para leer los datos de los distintos measurements (similar a SQL), la dirección *localhost* y puerto 8086 por defecto de InxfluxDB, el nombre de la base de datos *tesis*, username *root* y password *root* que no puede verse por razones de seguridad.

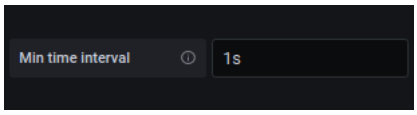


Figura 6.13: Configuración del intervalo de tiempo.

También es necesario configurar el intervalo de tiempo mínimo, esta es una variable utilizada por Grafana para agrupar los intervalos de tiempo que se leen desde la base de datos, donde es recomendable establecer en un valor aproximado a la frecuencia en la que se escribe la información en ella. En este caso, es necesario establecerla en 1 segundo ya que los datos se escriben constantemente en tiempo real.

Una vez configurada la conexión, será posible comenzar a visualizar e interactuar con las métricas de los paneles que han sido configurados para el desarrollo de este trabajo, donde primero es necesario seleccionar un dashboard como a continuación.

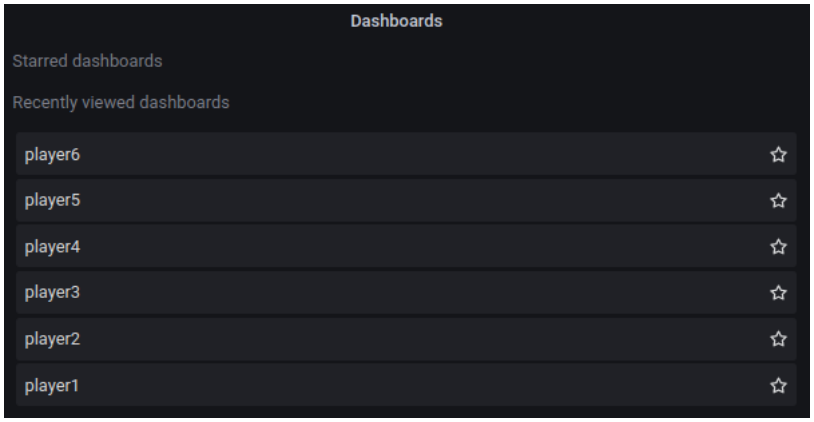


Figura 6.14: Dashboards disponibles para visualizar los paneles con las métricas.

En este caso se han configurado seis dashboards cada uno con un número de jugador distinto, por ejemplo de un equipo de fútbol. Cada dashboard a su vez fue configurado para mostrar exclusivamente las métricas de un determinado dispositivo que ha estado usando el jugador, esto quiere decir que el player1 ha estado utilizando el device1, el player2 el device2, y así sucesivamente.

Una vez seleccionado el dashboard para ver las métricas del jugador de interés, se redirigirá al usuario a una vista donde puede ver todos los paneles que fueron configurados cada uno con una métrica distinta. Por ejemplo si un entrenador deseara visualizar el rendimiento del jugador1 verá algo como las siguientes imágenes.

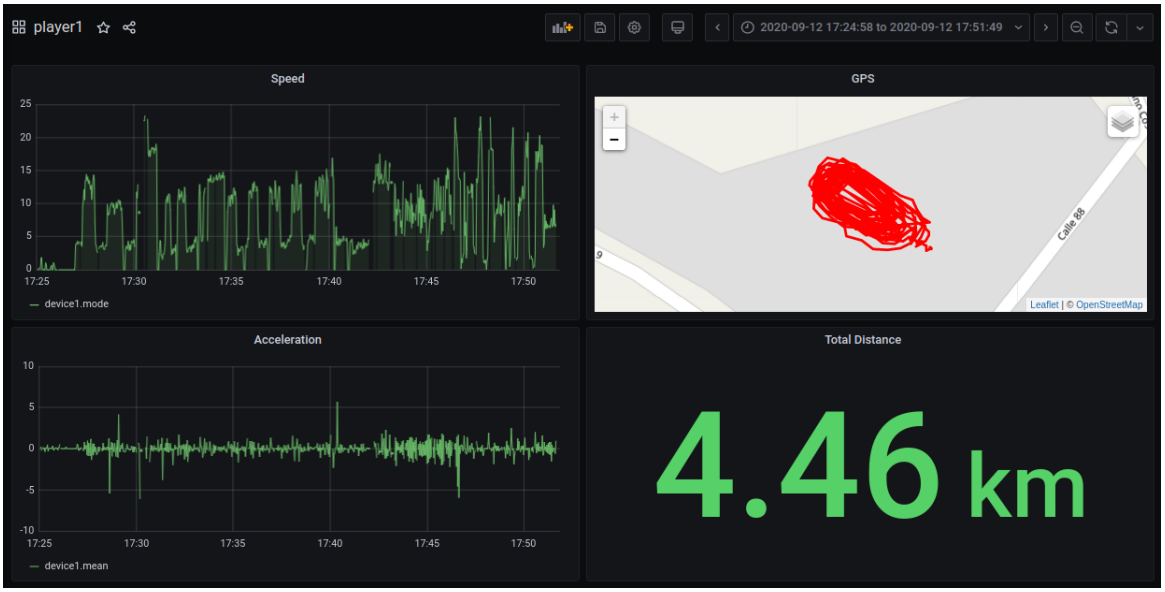


Figura 6.15: Métricas del jugador 1.

Figura 6.16: Métricas del jugador 1.

Entonces las métricas resultantes que podrá ver el entrenador son: *velocidad*, *aceleración*, *distancia total*, *cantidad de sprints*, *duración total en sprint*, *promedio de velocidad*, *velocidad máxima* y *gps*, este último se ve "pintado" el camino por donde se estuvo moviendo el jugador, de forma que funciona como "trackeo" del movimiento en todo momento, y que incluso es posible visualizar para cada instante de tiempo en que zona del mapa se estuvo, por ejemplo puede verse en que zona del mapa estuvo el jugador cuando tuvo un pico de velocidad de

23km/h, o en que parte del mapa estuvo cuando tuvo una desaceleración mayor a 5m/s, y en base a esto sacar propias conclusiones y estrategias para el jugador.

Cabe destacar, que los paneles y métricas mostradas para el desarrollo de este trabajo fueron conseguidas por los tesistas tras numerosas pruebas, correcciones y ajustes en el parque de una quinta, y por lo tanto, se debe seleccionar el rango de fecha y hora en que se desea ver la información almacenada. En caso de que deseen visualizarse las métricas de un entrenamiento en tiempo real, se debe seleccionar en la misma pestaña el tiempo máximo en que se leerá la información almacenada, por ejemplo los últimos 15 minutos, 30 minutos, 1 hora, 6 horas u el tiempo que se desee. En este caso, puede verse que el entrenamiento se llevó a cabo aproximadamente desde las 17:25hs hasta las 17:50hs, es decir, 25 minutos de entrenamiento. El panel donde se realiza esta configuración es el siguiente.

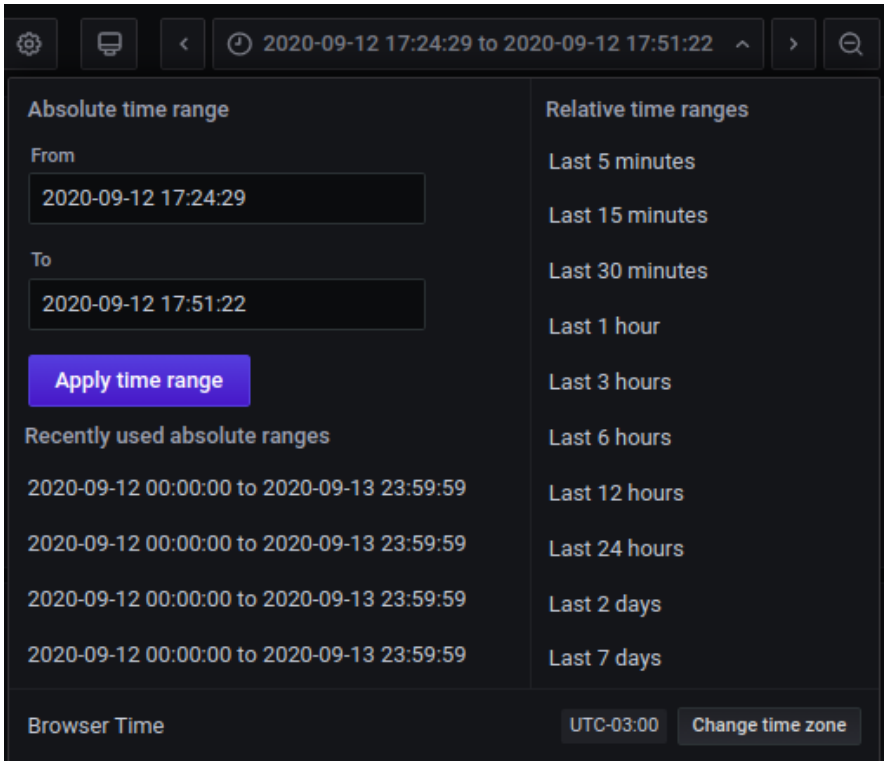


Figura 6.17: Rango de tiempo seleccionado para visualizar los datos del entrenamiento.

A su vez, en caso de estar visualizando las métricas en tiempo real, es necesario configurar la frecuencia de actualización del dashboard en general en el cual se actualizarán todos los paneles a medida que los nuevos datos se van almacenando,

de modo que los paneles con las métricas irán cambiando y alterándose en tiempo real en base a los datos que envíe el dispositivo.

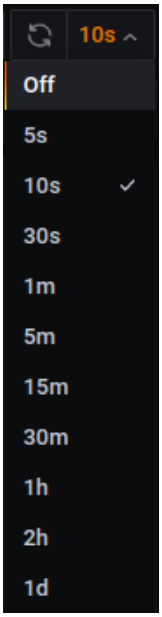


Figura 6.18: Frecuencia de actualización del dashboard player1.

Antes de explicar en detalle cada panel por separado junto con su configuración, se hará un breve resumen recordando desde dónde se consigue cada dato y porqué se están procesando de la forma en que se explicó anteriormente. Los mismos son:

1. Datos procesados por el dispositivo:
   - Velocidad por IMU.
   - Aceleración por IMU.
   - Distancia por GPS.
2. Datos procesados por el script python luego de leer los topics del broker MQTT:
   - Cantidad de sprints, basados en detecciones de picos en la velocidad.
   - Duración de cada sprint, calculada mediante el clock del procesador.
3. Datos procesados por Grafana:
   - Velocidad Máxima.

- Velocidad Promedio.
- Duración total de los sprints.
- Trazado de coordenadas en el mapa.

Debido a la naturaleza de la aplicación y su procesamiento de datos en tiempo real, no fue posible calcular algunas métricas como la cantidad de sprints y la duración de los mismos directamente mediante queries sobre InfluxDB o sobre Grafana. Realizar estos cálculos en tiempo real y solo basándose en el lenguaje InfluxQL era algo innecesario y que podía resolverse en el flujo de trabajo inmediato anterior, es decir en el procesamiento de los datos en el script de Python. Esto se debe a que con InfluxQL no se contaba con la manipulación de datos necesaria para determinar en tiempo real si un jugador entró en un sprint, así como también corroborrar que el mismo se mantenga en esa zona por un tiempo prolongado de manera precisa y que a su vez se almacene la cantidad de tiempo transcurrido en ese lapso. Esta problemática fue solucionada con el cálculo en tiempo real en el broker MQTT, procesando los datos en base al orden de su llegada como bien se demostró en el capítulo anterior en el código de Python donde se concluía que un jugador entró en sprint cuando sobrepasaba los 18km/hs y no bajaba de 14km/hs, y durante ese período de tiempo se utilizaba el clock (reloj) del procesador para calcular la cantidad de tiempo transcurrido que el jugador se mantuvo en esa zona.

Otra posible solución sería hacer los cálculos al finalizar el entrenamiento. Para calcular estas variables, se necesita tener bien delimitado el momento de inicio y fin de la acción, por eso como los datos llegan en tiempo real en una base de datos orientada a tiempo, es extremadamente complejo realizar consultas que permitan tener en cuenta el momento en el que llega el dato que delimita el fin de una acción, por esto que se optó por la solución en el código de Python. En resumen, si se realizaran las consultas y el procesamiento de los datos una vez finalizado el entrenamiento sería más fácil identificar estos rangos de tiempo, dando la posibilidad incluso de lograr otras métricas que tampoco han sido viables de calcular. Estas otras métricas que se decidió no incluir para la versión final del trabajo se basaban en dividir en más zonas las velocidades y las aceleraciones del jugador (puede decirse que ahora hay dos zonas, la zona de velocidad promedio y la zona de sprints), dando como resultado la posibilidad por ejemplo de calcular la cantidad de aceleraciones y desaceleraciones en zona 1 (2-3m/s), zona 2 (3-4m/s), zona 3 (4-5m/s) o zona 4

(mayor a 5m/s), aplicando la misma idea para la cantidad de tiempo transcurrido en zonas de velocidades, donde se podría tener un trackeo (seguimiento) más preciso de la performance de un jugador. Si bien el cálculo de estas métricas pudo haberse calculado en el script de Python, se decidió no incluirlas debido a que el cálculo de las mismas se basaría puramente en estructuras condicionales como if, $elif$, $else$ para delimitar las zonas y se concluyó que esa estrategia no era una solución lo suficientemente elegante para dejar plasmada en el desarrollo de un trabajo final como el presente.

Por consiguiente y volviendo a los paneles de Grafana, se mostrará en más detalle cada uno por separado junto con su configuración para mostrar cómo terminaron de obtenerse las métricas.

Figura 6.19: Métrica Velocidad.

Puede apreciarse que la velocidad almacenada del entrenamiento varía entre los 0km/h y y los 23km/h lo que quiere decir que durante la sesión se ha estado corriendo a distintas velocidades. Puede decirse que en esa sesión el tesista caminó, trotó y corrió, así como también ha estado completamente quieto como puede verse al inicio del gráfico. Este panel muestra la velocidad en crudo que se recibe por parte de los dispositivos, y la query configurada en el panel es la siguiente.

```
SELECT mode("current_velocity") FROM "device1" WHERE ("topic" = 'imu')
    AND $timeFilter GROUP BY time($__interval) fill(null)
```

Es necesario por parte de InfluxDB que todas las queries estén configuradas con al menos una función de agregación. En este caso se utilizó la función *mode*() que

retornará el valor más reciente para un field key en base a un timestamp, es decir, para este caso retornará las velocidades precisamente como fueron almacenadas. A su vez, en cada query es necesario agregar la variable de Grafana $\$timeFilter$ que es la encargada de filtrar y parsear la fecha y hora seleccionada en formato *Unix* (formato utilizado por InfluxDB para almacenar los timestamps), así como también agrupar los datos con la función $time(\$\_\_interval)$ que agrupará los datos recibidos en este caso por segundo ya que fue el intervalo de tiempo mínimo que se definió anteriormente, y por último la función $fill(null)$ para especificar que en los instantes de tiempo donde no se guardaron datos se rellenen con null, ya que de otra forma los resultados finales podrían verse afectados por datos basura (en caso de rellenar los espacios vacíos con, por ejemplo, el dato inmediato anterior, o cero).

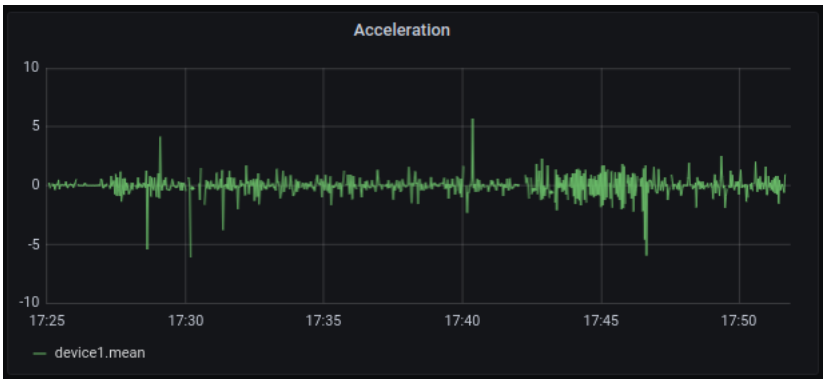


Figura 6.20: Métrica Aceleración.

En este gráfico puede apreciarse la aceleración que sufrió el dispositivo durante el entrenamiento. La aceleración puede ser negativa, en caso de que se haya desacelerado, o positiva, en caso de haber acelerado. Los picos máximos y mínimos son -5m/s y +5m/s aproximadamente, y puede compararse directamente con el gráfico de velocidad, es decir, puede verse que los picos de aceleración y desaceleración han sido cuando se elevó o redujo la velocidad. De todas formas, es importante remarcar que en esta sesión de entrenamiento no se realizaron movimientos de cambios de dirección, dando como resultado que las aceleraciones y desaceleraciones no hayan sido lo suficientemente fuertes como para superar por ejemplo los 10m/s como en el caso de que sí se hubiesen realizado cambios repentinos de dirección o movimientos anaeróbicos. Puede decirse que durante la sesión de entrenamiento de realizaron movimientos progresivos aumentando y reduciendo la velocidad de manera gradual para evitar lesiones. La query es la siguiente, donde se utilizó la

misma función *mode*() previamente explicada, la variable $*timeFilter* para aplicar el rango de tiempo seleccionado, la función *time*($*__interval*) para agrupar los datos por segundo (misma frecuencia de escritura) y la función *fill*(*null*) para mantener en null los instantes de tiempos donde no se guardó información.

```
SELECT mode("current_acceleration") FROM "device1" WHERE ("topic" = '
    imu') AND $timeFilter GROUP BY time($__interval) fill(null)
```

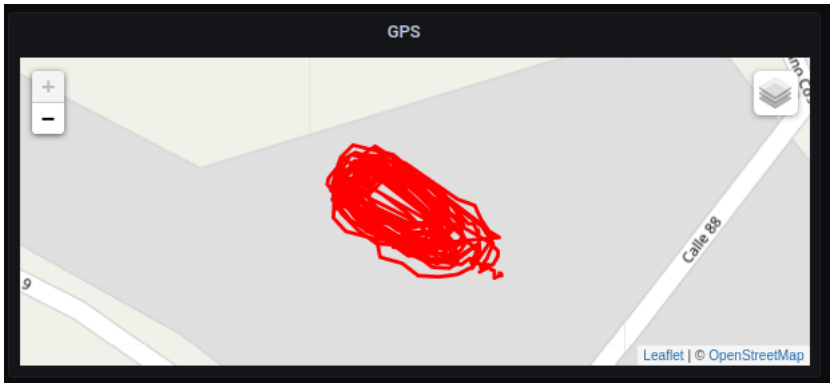


Figura 6.21: Métrica Localización GPS.

Este panel es el encargado de trackear el recorrido del jugador en todo momento mediante la latitud y longitud obtenida del GPS y por consiguiente "pintar" el camino por donde este estuvo moviéndose. Este trabajo lo hace pintando la distancia entre cada punto o par de coordenadas recibidas. Para mostrar este panel se utilizó el plugin *TrackMap* de Grafana, donde también es posible hacer y deshacer zoom en el mapa de manera interactiva. A su vez, como se ha mencionado en la explicación de la métrica de velocidad, es posible posicionar el mouse en cualquier punto de ese gráfico y se verá reflejado en el panel del GPS, mostrando el lugar en el que estuvo el jugador por ejemplo en los picos de velocidad o también para el gráfico de aceleración, donde se accede a la misma característica pero en ese caso para ver por ejemplo los picos de aceleraciones y desaceleraciones. Para remarcar, se comenta que para esta característica se escogió el plugin TrackMap ya que utiliza OpenStreetMap, uno de los proyectos colaborativos de información geográfica más grandes y con mayor aportes de la comunidad del mundo. Para el desarrollo de este trabajo, se realizó la sesión de entrenamiento en un parque de aproximadamente 10x50 metros y corriendo en círculos, dando por resultado el panel que se muestra

en la imagen. La query necesaria para su funcionamiento es la siguiente, donde se utilizan algunas funciones ya explicadas.

```
SELECT mean("latitude"), mean("longitude") FROM "device1" WHERE ("topic" = 'gps') AND $timeFilter GROUP BY time($__interval) fill(null)
```

Figura 6.22: Métrica Distancia total.

En este panel se visualiza la distancia total recorrida medida en kilómetros. Para ello, se deben sumar los valores recolectados del field key *distance* previamente calculados por el GPS, que es la distancia entre dos puntos por cada pull de datos que realiza el mismo. Debido a eso, está métrica se basa en sumar el total de todas las distancias almacenadas. Para ello y debido a la naturaleza de la métrica donde no es necesario utilizar ningún gráfico, se ha utilizado un panel de estadísticas o *StatPanel*, donde crear queries con este panel requiere aplicar una función de reducción en conjunto con una función de agregación como ya se venía utilizando. El trabajo encargado de una función de reducción es simplemente reducir N valores en un solo valor que será mostrado en el panel. La función que se utilizó es *total*(), que lo que hace es sumarizar todos los valores almacenados en este caso del field key *distance* y por consiguiente mostrar el valor resultante en el panel. Esta función de reducción se configura dentro del panel y no como una query de InfluxQL, es por ello que no aparece en la misma.

```
SELECT mode("distance") FROM "device1" WHERE ("topic" = 'gps') AND $timeFilter GROUP BY time($__interval) fill(null)
```

Figura 6.23: Métrica Cantidad de sprints.

Esta métrica se basa en mostrar la cantidad de sprints que se realizaron durante el entrenamiento, en este caso se realizaron 8 sprints. Para recordar, se dijo que se considera un sprint si el jugador o deportista excede una velocidad de 18km/h, esto quiere decir que el jugador ha entrado en zona de sprints y se mantendrá en la misma hasta que baje de 14km/h que fue la velocidad delimitada como cota inferior para demarcar la finalización del mismo. Es decir, una vez que el jugador supere los 18km/h sin importar el tiempo que sea que tarde en bajar de los 14km/h, se contará que realizó un sprint debido al esfuerzo que conlleva llegar a esa velocidad. Este trabajo se realiza en el script de Python, por lo que el trabajo encargado de este panel es sumar el total de veces que el script contó y registró un sprint en la base de datos. Para ello, en la query creada se utiliza la función de agregación *sum*() que suma la cantidad de veces que se registró un sprint, es decir, la cantidad de veces que se escribió un 1 en el field key *cant_sprints*. A su vez, se aplicó la función de reducción necesaria de los paneles de estadísticas, en este caso al igual que el anterior, se aplicó la función *total*() dentro de la configuración del panel.

```
SELECT sum("cant_sprints") FROM "device1" WHERE ("topic" = 'imu') AND
    $timeFilter GROUP BY time($__interval) fill(null)
```

Figura 6.24: Métrica Duración total de los sprints.

Esta métrica está sumamente relacionada con la anterior, el script de Python calcula la cantidad de sprints en base a si el jugador superó los 18km/h y se considera que el sprint finalizó si el jugador baja de los 14km/h. En ese período de tiempo, como se comentó en la explicación del código de Python en el capítulo 6, se sumariza el lapso de tiempo transcurrido en que el jugador entró en esa zona a través del uso del clock del procesador mediante la librería *timeit*, contando la cantidad de segundos transcurridos mientras el jugador está en esa zona para que, al finalizar el mismo, lo almacene en la base de datos junto con la duración calculada por el clock. Por consiguiente, el labor de este panel es totalizar la cantidad de tiempo que fue almacenado en *sprints_duration* mostrándolo en minutos. La query resultante es idéntica a la del panel anterior, se utiliza la función de agregación *sum*() en conjunto con la función de reducción *total*().

```
1 SELECT sum("sprints_duration") FROM "device1" WHERE ("topic" = 'imu')
    AND $timeFilter GROUP BY time($__interval) fill(null)
```

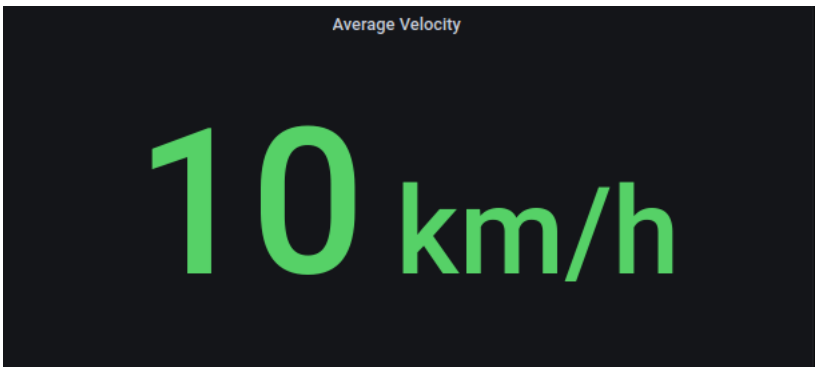


Figura 6.25: Métrica Promedio de velocidad.

Este panel simplemente hace un promedio de todas las velocidades almacenadas en el rango de tiempo seleccionado como también dinámicamente en caso de que las métricas del jugador se estén visualizando en tiempo real. El promedio de velocidad se visualiza en km/h. En la query de esta métrica, se utiliza la función de agregación *mean*() que computa el valor promedio de los valores no nulos almacenados en este caso dentro del field key *current_velocity*, en conjunto con la misma función de reducción *mean*() que no altera el resultado pero es requerida por el propio panel de estadística.

```
SELECT mean("current_velocity") FROM "device1" WHERE ("topic" = 'imu')
    AND $timeFilter GROUP BY time($__interval) fill(null)
```

Figura 6.26: Métrica Velocidad máxima.

Para finalizar, se muestra en este último panel la velocidad máxima a la que llegó el jugador durante el entrenamiento, también en km/h. Para calcular la velocidad máxima se utilizó la función de agregación *mode*() junto con la función de reducción *max*(), que calcula el valor máximo dentro de un rango de valores, en este caso también del field key *current_velocity*.

```
SELECT mode("current_velocity") FROM "device1" WHERE ("topic" = 'imu')
    AND $timeFilter GROUP BY time($__interval) fill(null)
```

Es importante agregar que se ha creado un usuario especial dentro de Grafana que será de solo lectura, para el caso en que el personal que utilice el sistema no sea un usuario técnico y por consecuente no modifique de manera no intencionada alguno

de los paneles de métricas o las queries configuradas para cada uno de estos. Este usuario es el usuario *Trainer* y tendrá acceso a todos los dashboards (uno cada jugador), a todas las métricas y acceso al panel de selección del rango de tiempo deseado para visualizar las métricas pasadas así como también para visualizarlas en tiempo real durante una sesión de entrenamiento.

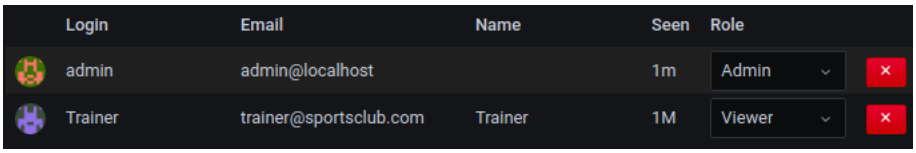


Figura 6.27: Usuarios creados para controlar Grafana.

Como adicional, se muestra como exportar los datos de cada panel a un archivo con formato CSV, dando al usuario la posibilidad de manipularlos a elección. Una librería muy utilizada y flexible para este tipo de operaciones es *Pandas*, que provee funciones de alto nivel para la manipulación y análisis de datos estructurados como lo es un CSV.

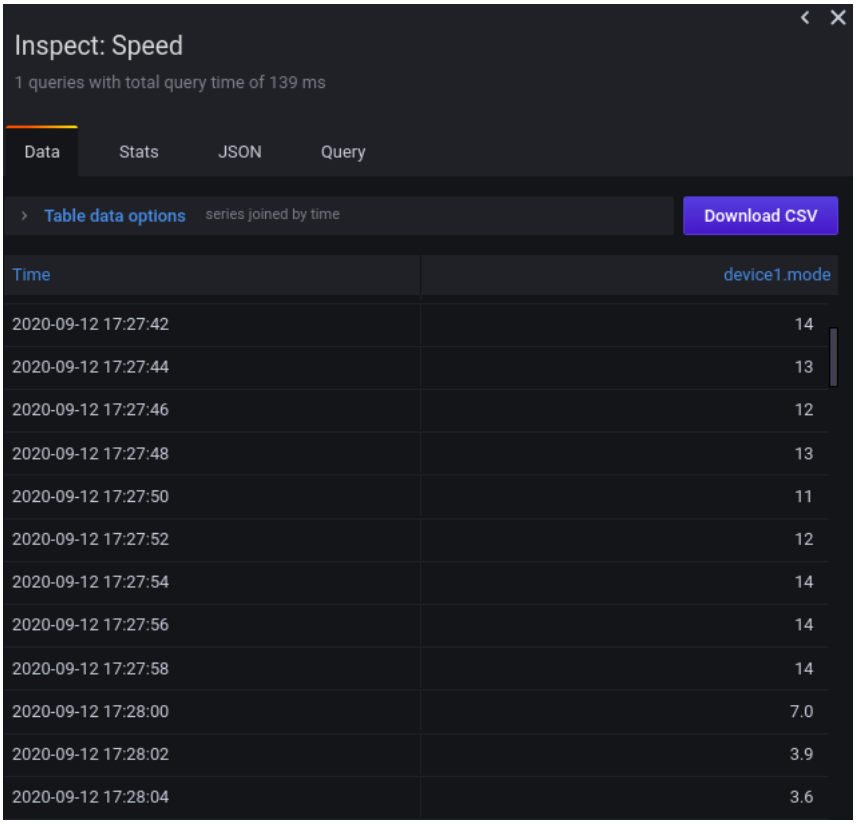


Figura 6.28: Exportando los datos del panel de velocidad a CSV.

	A	B
1	"Time"	"device1.speed"
2	2020-09-12 17:26:58	0.1738961
3	2020-09-12 17:27:00	3.5918067
4	2020-09-12 17:27:02	3.9657985
5	2020-09-12 17:27:04	3.8793258
6	2020-09-12 17:27:06	4.3485829
7	2020-09-12 17:27:08	4.0078671
8	2020-09-12 17:27:10	3.9482183
9	2020-09-12 17:27:12	4.2610836
10	2020-09-12 17:27:14	4.2424355
11	2020-09-12 17:27:16	3.7986993
12	2020-09-12 17:27:18	4.2325161
13	2020-09-12 17:27:20	3.5349074
14	2020-09-12 17:27:22	9.5660282
15	2020-09-12 17:27:24	10.373072
16	2020-09-12 17:27:26	12.151189
17	2020-09-12 17:27:28	10.569105
18	2020-09-12 17:27:30	13.390781
19	2020-09-12 17:27:32	13.755018
20	2020-09-12 17:27:34	14.410883
21	2020-09-12 17:27:36	13.475115
22	2020-09-12 17:27:38	14.007094
23	2020-09-12 17:27:40	13.028741
24	2020-09-12 17:27:42	13.550732
25	2020-09-12 17:27:44	13.135647
26	2020-09-12 17:27:46	12.113989
27	2020-09-12 17:27:48	12.769743
28	2020-09-12 17:27:50	10.642741
29	2020-09-12 17:27:52	12.020657
30	2020-09-12 17:27:54	13.788875
31	2020-09-12 17:27:56	14.343288
32	2020-09-12 17:27:58	13.554635

Figura 6.29: Leyendo CSV con los datos de velocidad del jugador 1.

Bajo esta explicación, se da por finalizada la explicación de las métricas logradas tras el procesamiento de información en los distintos nodos del flujo de trabajo:

1. Adafruit Feather Huzzah como Dispositivo Microcontrolador.
2. Broker MQTT en Raspberry Pi 4.
3. Script Python en Raspberry Pi 4.
4. Grafana en Raspberry Pi 4.

6.1.5. Raspberry Pi 4 como nodo middleware de almacenamiento y procesamiento

Para el desarrollo de este trabajo se ha utilizado una Raspberry Pi 4 con el fin de utilizarla como nodo intermediario de procesamiento y almacenamiento de los datos enviados por el dispositivo. A su vez, la RPi 4 será la encargada de correr todos los servicios necesarios para el completo funcionamiento de este trabajo. Entre

los servicios que tiene como responsabilidad ejecutar adecuadamente se encuentran *Mosquitto* (broker MQTT), *InfluxDB*, *Grafana*, *Script Python* para obtener los datos desde el broker y almacenarlos en InfluxDB.

También, la RPi 4 tendrá la responsabilidad de funcionar como *Hotspot*, esto quiere decir que creará una "red interna" para ofrecer cobertura WiFi con el objetivo de que los dispositivos llevados por los deportistas se conecten a ella y enviar los datos a la RPi 4. Este proceso se ha logrado configurando la RPi 4 en modo hotspot con una IP estática. Gracias a tener preconfigurada una IP estática, el código que será escrito en todos los dispositivos tendrán siempre como IP destino la misma IP que será la de la RPi 4 al cual enviarán todos los datos calculados por sus sensores.

Para llevar a cabo lo anteriormente mencionado se ha realizado un pequeño script que se ejecutará ni bien se encienda la RPI 4, donde su responsabilidad será la de lanzar correctamente los servicios anteriores junto con un manejo de excepciones en caso de que un servicio falle y tenga que ser levantado nuevamente. En otras palabras, este script será quien mantenga en funcionamiento la aplicación completa, y será la responsable del funcionamiento de la misma junto con el control de flujo y las medidas de seguridad necesarias para que la aplicación funcione correctamente durante toda la sesión de entrenamiento, donde deberá estar encendida en todo momento y recuperarse por si misma ante cualquier error.

Gracias a esto, cualquier persona interesada puede conectarse a la red creada por la RPi 4 y acceder mediante un navegador web a la dirección estática configurada, por ejemplo 192.168.1.100 más el puerto en el que corre el Grafana que es el 3000. Entonces, quien desee ver las métricas en Grafana sea en tiempo real o no, puede ingresar a la dirección 192.168.1.100:3000 y podrá elegir entre los dashboards disponibles para visualizar las métricas conseguidas de la sesión del atleta. Cabe mencionar, que la RPi 4 debe estar localizada en un punto intermedio del entrenamiento para que los dispositivos nunca pierdan la conexión WiFi debido al alcance y puedan continuar enviando los datos. Se calcula (en base a las pruebas que se han realizado sobre la RPi 4 en exteriores) que el alcance eficiente es de aproximadamente 50 metros, con lo cual la sesión debería realizarse en lo posible en un radio de 50 metros dejando RPi 4 en un punto medio. Para finalizar, se menciona que también es posible instalar repetidores alrededor de la cancha o predio de entrenamiento que sirvan para repetir u amplificar la señal o incluso instalar una antena

de mayor potencia en la RPi 4 para lograr un mayor alcance y extender el área de entrenamiento eficiente [85].

6.1.6. Comparativa con dispositivo profesional PlayerTek

En esta sección se llevará a cabo una breve comparativa entre el dispositivo diseñado para este trabajo y un dispositivo profesional utilizado por gran cantidad de equipos de primer nivel y de diversas disciplinas a nivel mundial. La comparativa se realizará con el dispositivo PlaterTek de Catapult.

Catapult es una empresa Australiana enfocada en construir y mejorar el rendimiento de los atletas y equipos en todos los niveles del deporte. Es líder mundial en lo que respecta a tecnología deportiva y posee soluciones para cada elemento del ecosistema deportivo, desde gestión de deportistas u atletas hasta análisis de video gracias a las métricas resultantes de los entrenamientos. Desde sus comienzos, Catapult ha desarrollado productos de esta índole enfocados en objetivos simulares, pero entre los que más se destacan son el PlayerTek y el PLAYR. La principales diferencia entre éstos radica en que PlayerTek es un producto más accesible por el mercado de consumo debido a su inferior precio, pero a coste de brindarnos menos cantidad de métricas. PLAYR, es un producto más completo, brinda mayor cantidad de métricas con mayor precisión y el más usado por los deportistas y equipos de elite, pero también tiene un costo mayor. En este caso estaremos hablando específicamente del PlayerTek ya que es el que pudo utilizarse y estudiar para realizar una breve comparativa. Cabe destacar, que la versión de PlayerTek que se utilizó para realizar la comparativa sólo trabaja de manera offline, es decir, se entrena con el dispositivo y al finalizar, es necesario sincronizarlo con la aplicación móvil para descargar los datos.

Para comenzar, se mostrará la ficha técnica de las especificaciones de hardware del dispositivo PlayerTek para tener como referencia sus capacidades en comparación con el dispositivo de este trabajo [86].

HARDWARE	
DIMENSIONS	84mm x 42mm x 21mm
WEIGHT	42g
BATTERY	7hrs
GLOBAL POSITIONING	10Hz GPS/GNSS
INERTIAL SAMPLING RATE	400HZ (recorded at 100Hz)
LIVE DATA	High Performance Cortex-M4 CPU for Live Data calculations*
ACCELEROMETER	Accurate +/-18g accelerometer samples at 400Hz in 3 axes
HEART RATE	Polar H1 Compatible*
WIRELESS COMMUNICATION	Separate Bluetooth low energy chipset + 2.4GHz antenna
WIRELESS RANGE	Up to 100 metres with dedicated M0 CPU for reliable communication
PLAYERTEK CLOUD ANALYTICS	Advanced (inc Zonal + Acute Chronic)*

Figura 6.30: Ficha técnica del dispositivo PlayerTek.

Las métricas que nos otorga este dispositivo son: *distancia*, *velocidad máxima*, *distancia en sprint* y *cantidad de sprints*, y puede apreciarse en la siguiente imagen.

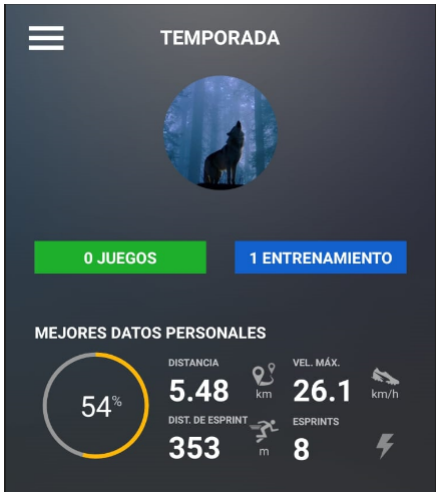


Figura 6.31: Métricas del dispositivo PlayerTek.

La aplicación de PlayerTek a su vez muestra un gráfico con las métricas de los últimos 10 partidos (u entrenamientos), que en este caso como sólo se ha realizado una sesión, esta última será quien tendrá los valores máximos.

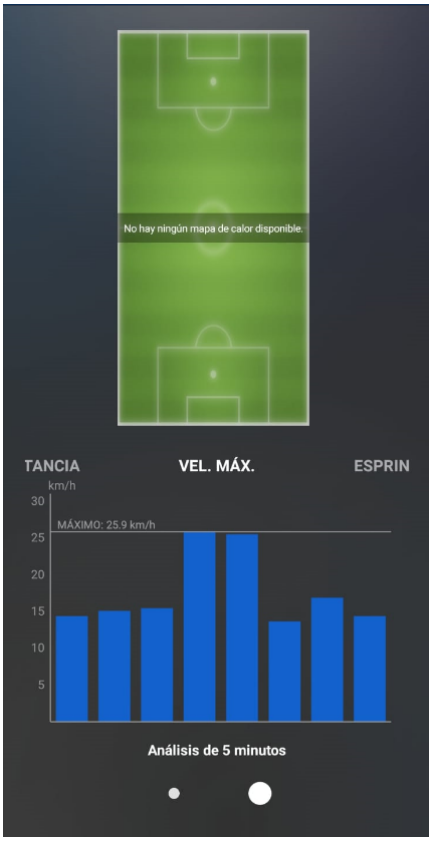


Figura 6.32: Métricas del dispositivo PlayerTek.

En esta imagen se puede apreciar la también el gráfico de la velocidad máxima. También, se ve el dibujo de una cancha de fútbol donde se dibujaría en color las zonas en las que se realizó el entrenamiento. Para ello, es necesario una vez terminada la sesión de entrenamiento, abrir la aplicación y marcar mediante la integración con Google Maps el rectángulo de la cancha o directamente la zona donde se ha realizado el entrenamiento. Para este trabajo, esa característica no se ha configurado ya que la zona donde se realizó el entrenamiento es no es una cancha de fútbol ni un espacio reducido que pueda marcarse desde la aplicación, el entrenamiento se realizó siguiendo un recorrido al aire libre y no es de utilidad para lo que respecta este trabajo. Igualmente, a continuación se deja una imagen donde puede verse como sería el comportamiento de esta característica.

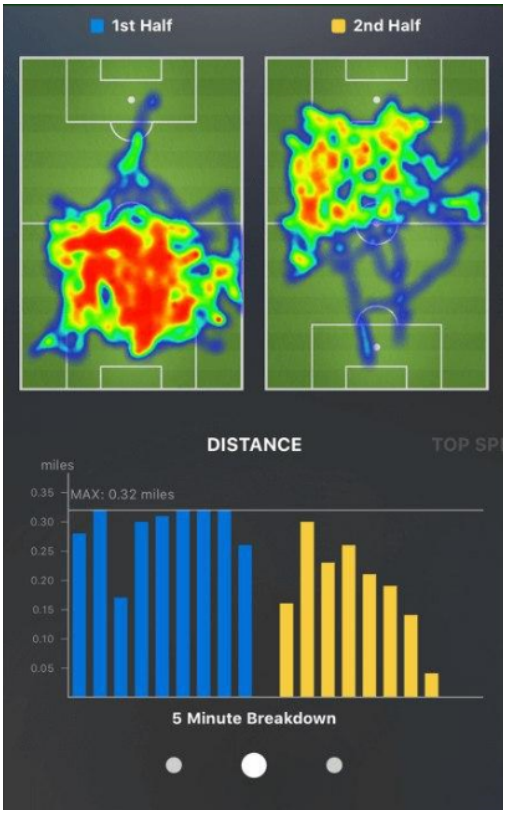


Figura 6.33: Heatmap de un jugador de fútbol utilizando el dispositivo PlayerTek.

Como puede verse, la aplicación de PlayerTek nos brinda la posibilidad de visualizar las zonas por donde se corrió más y se estuvo una mayor cantidad de tiempo durante la sesión, siendo cuanto más oscuro el color, mayor cantidad de movimiento en ese sector y cuanto más suave el color, menos tiempo [87].

Estas son las principales características que nos permite estudiar el dipositivo PlayerTek de Catapult en la versión donde se obtienen las métricas de modo offline y deben volcarse los datos luego de cada sesión de entrenamiento o partido. Se adiciona, que la versión PlayerTek+ ya incorpora esta característica, y obviamente también el dispositivo PLAYR.

Entonces, volviendo al capítulo anterior donde se demuestra el funcionamiento del dispositivo desarrollado para este trabajo, puede notarse que el dispositivo desarrollado por los tesistas iguala en cantidad de métricas conseguidas e incorpora otras que sólo podrían conseguirse adquiriendo la versión más completa de Catapult, con a su vez la gran ventaja de poder visualizar los gráficos de las métricas en tiempo

real durante los entrenamientos y/o partidos.

6.1.6.1. Pricing

A continuación se armará una breve lista de los precios de los productos utilizados para el desarrollo del presente trabajo y también del dispositivo PlayerTek. Todos los precios han sido consultados desde el sitio *amazon.com* en la fecha 21/07/2020.

Dispositivo tesistas:

- Ublox-NEO-M8N: 20 USD
- MPU9250: 4 USD
- Adafruit Feather HUZZAH ESP8266: 10 USD

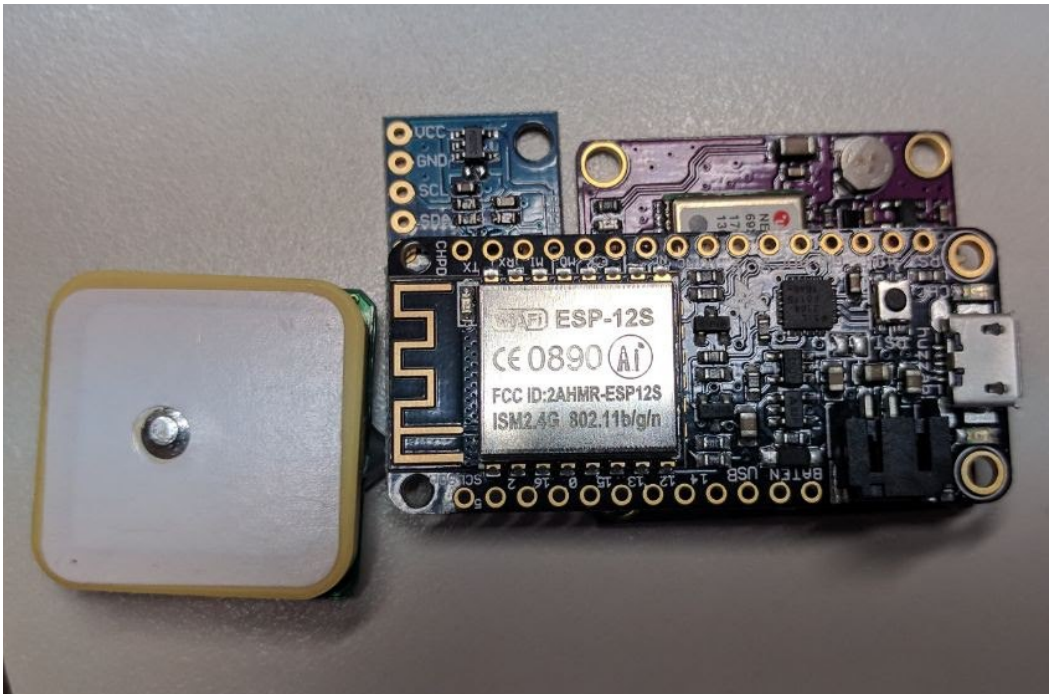


Figura 6.34: Componentes del dispositivo de los tesistas.

Dado que los dispositivos de Catapult están en constante evolución, al día de hoy han lanzado al mercado un nuevo dispositivo PlayerTek, este es el PlayerTek+ y

reemplaza a su predecesor, introduciendo como principal característica la visualización en tiempo real de las métricas. No se ha podido encontrar el valor actual del PlayerTek original que fue utilizado al realizar esta comparativa, pero se calcula que el precio es tan solo un poco menor que el PlayerTek+.

Dispositivo Catapult:

- PlayerTek+: 200 USD

Figura 6.35: Dispositivo Catapult PlayerTek.

Puede notarse que el precio del dispositivo creado para este trabajo es ampliamente menor que el de PlayerTek (34 USD contra 200 USD), y que a su vez, no solo incorpora las mismas funcionalidades y métricas, sino que también como ha podido verse, agrega otras métricas y características que sólo podrían encontrarse en el dispositivo de cabecera de PlayerTek.

Capítulo 7

Conclusiones y líneas de trabajo futuro

En este capítulo se expone un resumen de los objetivos logrados durante el desarrollo de la tesina. Por último, se detallan los posibles trabajos futuros que se desprenden de esta tesis, teniendo también en cuenta las problemáticas encontradas.

7.1. Conclusiones

A lo largo de la tesina se ha desarrollado y explicado una primera versión en lo que respecta a un sistema de *Motion Tracking* para deportistas y/o aficionados del deporte en tiempo real, cubriendo una gran cantidad de conceptos en materia de *Internet de las Cosas*(IoT). Se decidió incursionar en esta área debido al crecimiento exponencial que están teniendo este tipo de dispositivos que tienen la habilidad de conectarse a internet, creando un inmenso mundo de posibilidades, alternativas y aplicaciones. A su vez, se escogió aplicar IoT específicamente al área del deporte a causa de la notoria escasez de alternativas *open source* junto al elevado valor monetario de los dispositivos privativos disponibles en el mercado, lo que nos motivó a sumergirnos en esta evolutiva, desafiante y competitiva área con la finalidad de erigir un sistema íntegro, de código libre y al alcance de todos los cautivos por la materia. Hemos alcanzado un prototipo funcional, objeto de investigación y

mejora, con el que pudimos realizar pruebas suficientes para determinar que incluso con componentes de hardware económicos es posible obtener valiosos resultados que ayudarán a incrementar, tras su análisis, la performance y rendimiento de los deportistas.

Para el desarrollo de este trabajo, se comenzaron añadiendo los sensores (GPS, acelerómetro, giroscopio y magnetómetro) a la placa de desarrollo para en su conjunto recopilar y procesar los datos provenientes de los mismos. Para el procesamiento de los datos en crudo se utilizaron operaciones matemáticas complejas que permitieron obtener los valores de velocidad y aceleración del deportista. Debido a que los sensores tienden a poseer errores de drift, fue necesario investigar exhaustivamente distintos filtros, algoritmos y formas de integrar los datos correctamente para alcanzar resultados reales y precisos. Estos métodos son soluciones que proponen la colaboración del acelerómetro, magnetómetro, giroscopio y GPS para realizar estimaciones significativamente más asertivas. Otras complicaciones provienen de la dificultad en detectar ciertos valores erróneos dentro de los rangos de operación normal, ya que no pueden diferenciarse de los valores normales como los valores extremos fuera de rango.

También fue necesario manejar la lógica del dispositivo para controlar la conexión, reconexión y el envío de los datos mediante el protocolo MQTT hacia el broker configurado a través de canales definidos para cada conjunto de datos. Para esto, se configuró una RPi 4 como broker y nodo central de procesamiento y almacenamiento, donde se ejecuta un script Python que lee los datos en tiempo real de los canales del broker, los procesa para obtener métricas y les da el formato adecuado para ser almacenados en InfluxDB. Por último, se crearon en Grafana los dashboards vinculados a cada dispositivo conectado con las métricas de cada jugador. Estas métricas están distribuidas en diversos paneles con el objetivo de monitorear los resultados y por ende mejorar la performance de los deportistas, facilitando el análisis, estrategia y toma de decisiones de los entrenadores en tiempos donde el deporte tiene una presencia cada vez más competitiva.

Los algoritmos utilizados en este desarrollo fueron los que produjeron mejores resultados en base al testeo realizado. Es posible que en otras condiciones de prueba más exhaustivas generen resultados que representen de manera más precisa la actividad real. Es crítico evaluar la validez y fiabilidad de múltiples algoritmos de fusión de

sensores durante un entrenamiento real, obteniendo información de distintos patrones e intensidad de movimiento. Se necesitan amplios conocimientos de física para aplicar los filtros de procesamiento de las señales crudas obtenidas y lograr un correcto procesamiento de los de datos. A su vez, se necesita analizar la forma correcta de aplicar estos filtros a los datos en tiempo real, ya que la mayoría de las soluciones encontradas son aplicables solo a Datasets preexistentes.

Para finalizar, se concluye que hemos alcanzado el objetivo planteado, y que como primer prototipo hemos logrado una aproximación a lo que se denomina *Motion Tracking*. El dispositivo servirá como base para realizar un producto aún más fiable, más preciso y más competitivo que genere un impacto a mayor escala en lo que respecta al estudio y análisis del área del deporte beneficiándose de la creciente demanda de la tecnología IoT.

7.2. Trabajos futuros

A continuación, se describen los posibles trabajos futuros que se desprenden de esta tesina:

- Investigar microcontroladores multinúcleo que sean capaces de paralelizar en hilos los módulos de polling de GPS, del IMU y de recolección WiFi, para poder calcular de manera precisa la integral en el tiempo transcurrido de muestreo. A su vez y mediante el uso de microcontroladores multinúcleo, se podría modificar la integración para utilizar un tiempo constante, en lugar del tiempo transcurrido para obtener las muestras.
- Mejorar el procesamiento de datos crudos mediante la utilización de filtros, específicamente el filtro de Kalman, que permite la fusión, corrección y predicción de estados futuros de los datos de entrada. El mismo se utiliza para obtener un dato de ubicación aún más preciso con la posición en forma de coordenadas obtenida del GPS y el cuaternión del IMU.
- Investigar módulos de WiFi capaces de abarcar toda el área correspondiente a la cancha de fútbol, rugby, etc. para no perder conectividad. Notar que este

apartado también depende de la potencia del módulo WiFi del dispositivo configurado como hotspot.

- Investigar sobre la instalación de repetidores en puntos específicos de la cancha para abarcar el área total de conectividad de los dispositivos de manera eficiente.

- Escribir los datos procesados por el dispositivo en un módulo de memoria micro SD, para ser analizados posterior al entrenamiento. De esta manera, se podrían aplicar múltiples algoritmos/filtros a los datos crudos obtenidos para ser comparados y a su vez evitar la pérdida de datos por desconexión del dispositivo con el broker MQTT.

- Investigar la manera óptima de delimitar las zonas de velocidades en las que se encuentra el deportista en tiempo real para obtener una mayor cantidad de métricas.

- Crear una interfaz web para controlar los servicios de la RPi 4, es decir, iniciar y parar los servicios del broker Mosquitto, InfluxDB, Grafana y el script Python de manera interactiva junto con un panel de mensajes de errores de ejecución, como también otro panel para visualizar los logs que envía cada dispositivo durante el entrenamiento.

- Crear una aplicación específica, ya sea web y/o mobile, para manejar los parámetros de configuración de las métricas y visualizar las mismas de manera que se puedan ajustar a las necesidades de los expertos en el área del deporte sin tener que recurrir a un software de propósito general como Grafana.

Bibliografía

[1] Ahsan Ikram y col. «Approaching the Internet of things (IoT): A modelling, analysis and abstraction framework». En: *Concurrency and Computation: Practice and Experience* 27 (sep. de 2013).

[2] Chathuranga M. Wijerathna Basnayaka. «Internet of Things for Smart Cities». En: (abr. de 2020).

[3] Suwimon Vongsingthong y Sucha Smanchat. «Internet of Things: A review of applications and technologies». En: (ene. de 2014). DOI: 10.14456/sjst.2014.38.

[4] Sridip Paul y col. «An IoT Based Home Automation System». En: *IOP Conference Series: Materials Science and Engineering* 623 (oct. de 2019), pág. 012014. DOI: 10.1088/1757-899X/623/1/012014.

[5] Chen Yang, Weiming Shen y Xianbin Wang. «The Internet of Things in Manufacturing: Key Issues and Potential Applications». En: *IEEE Systems, Man, and Cybernetics Magazine* 4 (ene. de 2018), págs. 6-15. DOI: 10.1109/MSMC.2017.2702391.

[6] Robert Harmon, Enrique Castro-Leon y Sandhiprakash Bhide. «Smart cities and the Internet of Things». En: ago. de 2015, págs. 485-494. DOI: 10.1109/PICMET.2015.7273174.

[7] Nicolaie Fantana y col. «Internet of Things - Converging Technologies for Smart Environments and Integrated Ecosystems». En: ene. de 2013, págs. 153-204. ISBN: ISBN 978-87-92982-73-5 (print) ISBN 978-87-9282-96-4 (ebook).

[8] Kasey Miller, Bryan O'Halloran y Anthony Pollman. «Securing the Internet of Battlefield Things While Maintaining Value to the Warfighter». En: feb. de 2019.

[9] Vishal Gotarane y Sandeep Raskar. «IoT Practices in Military Applications». En: abr. de 2019, págs. 891-894. DOI: 10.1109/ICOEI.2019.8862559.

[10] Pavlina Kröckel, Alexander Piazza y Kathrin Neuhofer. «Dynamic Network Analysis of the Euro2016 Final: Preliminary Results». En: ago. de 2017. DOI: 10.1109/FiCloudW.2017.98.

[11] Kenneth Cortsen y Daniel Rascher. «The Application of Sports Technology and Sports Data for Commercial Purposes». En: nov. de 2018. ISBN: 978-1-78984-482-5. DOI: 10.5772/intechopen.80742.

[12] Pedro Gamardo. *Evaluación de La Aptitud Física-Motora del Futbolista Menor: Proceso de formación*. Oct. de 2011. ISBN: 978-3846567494.

[13] Daniel Castillo y col. «Comparación de las demandas de un juego reducido y un partido oficial en jugadores juveniles (sub-19) de élite». En: mar. de 2018.

[14] Robin Thorpe y col. «Monitoreo de la fatiga durante el período competitivo de una temporada en jugadores élite de fútbol». En: *International Journal of Sports Physiology and Performance* 10 (nov. de 2015), págs. 958-964.

[15] Ugur Acar y col. «Designing An IoT Cloud Solution for Aquaculture». En: jun. de 2019, págs. 1-6. DOI: 10.1109/GIOTS.2019.8766428.

[16] Stefan Nastic y col. «PatRICIA – A Novel Programming Model for IoT Applications on Cloud Platforms». En: dic. de 2013, págs. 53-60. DOI: 10.1109/SOCA.2013.48.

[17] Barry Haughian. *Design, Launch, and Scale IoT Services: A Practical Business Approach*. Apress, 2018. ISBN: 978-1-4842-3712-0. URL: http://gen.lib.rus.ec/book/index.php?md5=5a2bf7e542c8ff3ba6b0a88f86a15d13.

[18] A Olabisi, Oluwashola Adeniji y Abeng Enangha. «A Comparative Analysis of Latency, Jitter and Bandwidth of IPv6 Packets Using Flow Labels in Open Flow Switch in Software Defined Network». En: (jul. de 2019), págs. 30-36.

[19] Dominik Samociuk y Blazej Adamczyk. «Secure gateway for Internet of Things with internal AAA mechanism». En: *Theoretical and Applied Informatics* 28 (abr. de 2017), págs. 17-35. DOI: 10.20904/283017.

[20] Dr. Yusuf Perwej y col. «The Internet-of-Things (IoT) Security : A Technological Perspective and Review». En: Volume 5 (feb. de 2019), Page 462-482. DOI: 10.32628/CSEIT195193.

[21] electronicsforu. *Microcontroller Development Boards*. URL: https://www.electronicsforu.com/buyers-guides/hardware-buyers-guide/microcontroller-development-boards.

[22] Marshall Brain. *How Microcontrollers Work*. URL: https://electronics.howstuffworks.com/microcontroller1.html.

[23] Jonathan W. Valvano. «Embedded Systems - Shape The World». En: 2014.

[24] Jacob Fraden. *Electronics Handbook of Modern Sensors Physics Designs and Applications*. 3rd. Springer, 2003. ISBN: 0-387-00750-4,0-387-00750-4.

[25] Gabriela Quiroz Córdova. «Laboratorio Analógico». Tesis de mtría. The address of the publisher: Universidad de las Américas Puebla, 2007.

[26] GPS.gov. Página oficial de GPS del gobierno de los Estados Unidos. URL: https://www.gps.gov.

[27] Navigation Center. U.S. Coast Guard Navigation Center. URL: https://www.navcen.uscg.gov/?Do=constellationStatus.

[28] ESOA. Emea Stellite Operators Association. URL: https://www.esoa.net/technology/satellite-orbits.asp.

[29] INFORMATION, NAVIGATION ANALYSIS CENTER FOR POSITIONING y TIMING. URL: https://www.glonass-iac.ru/en/GPS/index.php.

[30] European Globan Navigation Setellite Systems Agency. URL: https://www.gsc-europa.eu/system-service-status/constellation-information.

[31] Gustavo Bento Mansur y Luiz Danilo Damasceno Ferreira. «ERROR BEHAVIOR OF ATOMIC CLOCKS ABOARD GPS SATELLITES». En: *Boletim de CiÃGeodÃ* 25 (de 2019). URL: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1982-21702019000400204&nrm=iso.

[32] Peter Larsson. «Global Positioning System and Sport-Specific Testing». En: *Sports medicine (Auckland, N.Z.)* 33 (feb. de 2003), págs. 1093-101. DOI: 10.2165/00007256-200333150-00002.

[33] Dinesh Sathyamoorthy y col. «Evaluation of the effect of global positioning system (GPS) satellite clock error via GPS simulation». En: *Defence ST Technical Bulletin* (abr. de 2015).

[34] Nyen Thin Li y col. «GPS Systems Literature: Inaccuracy Factors And Effective Solutions». En: *International journal of Computer Networks Communications* 8 (abr. de 2016).

[35] National Marine Electronics Association. URL: https://www.nmea.org/.

[36] National Marine Electronics Association. *NMEA 0183 Standard*. Version 4.11.

[37] u-Blox. *u-blox 8 (M8) Receiver Description Including Protocol Specification*. URL: www.u-blox.com.

[38] Sparkfun. *How to Select an Accelerometer*. URL: https://learn.sparkfun.com/tutorials/accelerometer-basics/how-to-select-an-accelerometer.

[39] Peter Corke. *Robotics, Vision and Control - Fundamental Algorithms in MATLAB®*. Vol. 73. Ene. de 2011. ISBN: 978-3-642-20143-1.

[40] Oliver J. Woodman. «An introduction to inertial navigation». En: 2007.

[41] Epson. *Gyro sensors - How they work and what's ahead*. URL: https://www5.epsondevice.com/en/information/technical_info/gyro/.

[42] Bonnie Baker. *Aplicar la fusión de sensores a acelerómetros y giroscopios*. URL: https://www.digikey.com/es/articles/apply-sensor-fusion-to-accelerometers-and-gyroscopes.

[43] Facundo Luis Palavecino. «Práctica Profesional Supervisada, Soluciones de IoT basada en sensores para controlar el entrenamiento deportivo de élite». En: (sep. de 2017).

[44] Invesense. *Product Specification*. URL: https://invensense.tdk.com/products/motion-tracking/9-axis/mpu-9250/.

[45] Adafruit. *Pinouts*. URL: https://learn.adafruit.com/adafruit-feather-huzzah-esp8266/pinouts.

[46] Mirjana Maksimovic y col. «Raspberry Pi as Internet of Things hardware: Performances and Constraints». En: jun. de 2014.

[47] Anand Nayyar y Vikram Puri. «Raspberry Pi-A Small, Powerful, Cost Effective and Efficient Form Factor Computer: A Review». En: *International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE)* 5 (dic. de 2015), págs. 720-737.

[48] Eclipse Foundation. *MQTT: The Standard for IoT Messaging*. URL: http://mqtt.org.

[49] Meena Singh y col. «Secure MQTT for Internet of Things (IoT)». En: abr. de 2015, págs. 746-751. DOI: 10.1109/CSNT.2015.16.

[50] Hamid Hasan y Bahaa Alhusainy. «Evaluation of MQTT Protocol for IoT Based Industrial Automation». En: 8 (dic. de 2018), págs. 19364-19369.

[51] Michael Yuan. *Getting to know MQTT*. URL: https://developer.ibm.com/articles/iot-mqtt-why-good-for-iot/.

[52] Arduino. *FAQ*. URL: https://www.arduino.cc/en/main/FAQ.

[53] Arduino. *Arduino Code*. URL: https://www.circuito.io/blog/arduino-code/.

[54] InfluxData. *Time series database (TSDB) explained*. URL: https://www.influxdata.com/time-series-database/.

[55] Muntazir Fadhel, Emil Sekerinski y Shucai Yao. «A Comparison of Time Series Databases for Storing Water Quality Data». En: abr. de 2019, págs. 302-313. ISBN: 978-3-319-89796-7. DOI: 10.1007/978-3-030-11434-3_33.

[56] Dmitry Namiot. «Time Series Databases». En: oct. de 2015.

[57] InfluxData. *InfluxDB concepts*. URL: https://docs.influxdata.com/influxdb/v1.8/concepts/.

[58] IETF. *Date and Time on the Internet: Timestamps*. URL: https://www.ietf.org/rfc/rfc3339.txt.

[59] Mohammad Abu Kausar y Mohammad Nasar. «Suitability Of Influxdb Database For Iot Applications». En: 10 (ago. de 2019), pág. 7.

[60] Grafana Labs. *What is Grafana?* URL: https://grafana.com/docs/grafana/latest/.

[61] J. Turnbull. *The Art of Monitoring*. James Turnbull, 2014. ISBN: 0988820242, 9780988820241. URL: http://gen.lib.rus.ec/book/index.php?md5=052da09ceb2b72825e6d67277171f2a5.

[62] Python Software Foundation. *Python Documentation*. URL: https://www.python.org/doc/.

[63] Bell Labs Bjarne Stroustrup. *C++ Documentation*. URL: https://isocpp.org/.

[64] Mikal Hart. *TinyGPS++*. URL: https://github.com/mikalhart/TinyGPSPlus.

[65] Sparkfun. *MPU-9250 9 DOF IMU Arduino Library*. URL: https://github.com/sparkfun/SparkFun_MPU-9250_Breakout_Arduino_Library.

[66] Nick O'Leary. *Arduino Client for MQTT*. URL: https://pubsubclient.knolleary.net/api.html.

[67] W3. *Gyroscope*. URL: https://www.w3.org/TR/motion-sensors/#gyroscope.

[68] Simone Ludwig y col. «Comparison of attitude and heading reference systems using foot mounted MIMU sensor data: basic, Madgwick, and Mahony». En: mar. de 2018, pág. 96.

[69] S. O. H. Madgwick, A. J. L. Harrison y R. Vaidyanathan. «Estimation of IMU and MARG orientation using a gradient descent algorithm». En: 2011.

[70] Yan-Bin Jia. «Quaternions and Rotations *». En: 2015.

[71] Sebastian O. H. Madgwick. «An efficient orientation filter for inertial and inertial / magnetic sensor arrays». En: 2010.

[72] JACK KUIPERS. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace and Virtual Reality*. Ene. de 2002. ISBN: 0691102988.

[73] W3. *Motion Sensors Explainer*. URL: https://www.w3.org/TR/motion-sensors/#linear-acceleration-sensor.

[74] Y. Yang, Yanfei Zhao y Dali Kang. «Integration on acceleration signals by adjusting with envelopes». En: 2016.

[75] Kurt Seifert y Oscar Camacho. «Implementing Positioning Algorithms Using Accelerometers». En: (2007).

[76] Ryo Kawaguchi y Masaki Bandai. «A Distributed MQTT Broker System for Location-based IoT Applications». En: ene. de 2019, págs. 1-4. DOI: 10.1109/ICCE.2019.8662069.

[77] Jaidip Kotak, A. Shah y P. Rajdev. «A Comparative Analysis on Security of MQTT Brokers». En: ene. de 2019, 7 (5 pp.)-7 (5 pp.) DOI: 10.1049/cp.2019.0180.

[78] Roger Light. «Mosquitto: server and client implementation of the MQTT protocol». En: *The Journal of Open Source Software* 2 (mayo de 2017). DOI: 10.21105/joss.00265.

[79] Eclipse Foundation. *Paho MQTT Documentation*. URL: https://www.eclipse.org/paho/.

[80] InfluxData. *influxdb-client-python*. URL: https://github.com/influxdata/influxdb-client-python.

[81] Léo Djaoui y col. «Maximal Sprinting Speed of Elite Soccer Players During Training And Matches». En: *Journal of strength and conditioning research* 31 (jun. de 2017), págs. 1509-1517. DOI: 10.1519/JSC.0000000000001642.

[82] Ana Delalija y Vesna Babić. «Reaction time and sprint results in athletics». En: *International Journal of Performance Analysis in Sport* 8 (jul. de 2008), págs. 67-75. DOI: 10.1080/24748668.2008.11868436.

[83] Neil Bezodis, Steffen Willwacher y Aki Salo. «The Biomechanics of the Track and Field Sprint Start: A Narrative Review». En: *Sports Medicine* 49 (jun. de 2019). DOI: 10.1007/s40279-019-01138-1.

[84] Rolf Graubner y Eberhard Nixdorf. «Biomechanical Analysis of the Sprint and Hurdles Events at the 2009 IAAF World Championships in Athletics». En: *IAAF World Championships* 26 (feb. de 2011), págs. 1-35. DOI: 10.1007/s40279-019-01138-1.

[85] Sumon Debnat y col. «Raspberry Pi Configuration for Access-Point and Its Throughput Measurements in IEEE 802.11n Wireless Networks». En: dic. de 2016.

[86] Catapult. *Catapult Playertek*. URL: https://www.catapultsports.com/products/playertek.

[87] Catapult. «A Guide to the PLAYERTEK software system». En: *Playertek Manual Guide* 1 (nov. de 2015), págs. 1-29.